Juha-Pekka Jokela

YLEISKÄYTTÖINEN GRAFIIKKAKIRJASTO

YLEISKÄYTTÖINEN GRAFIIKKAKIRJASTO

Juha-Pekka Jokela
Opinnäytetyö
Kevät 2013
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

ALKULAUSE

Tehty Helsingissä 2012-2013 Juha-Pekka Jokelan toimesta.

Juha-Pekka Jokela

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä(t): Juha-Pekka Jokela
Opinnäytetyön nimi: Yleiskäyttöinen grafiikkakirjasto
Työn ohjaaja(t): Esko Harvala
Työn valmistumislukukausi ja -vuosi: Kevät 2013
Sivumäärä: 18 + liitteet

Opinnäytetyö tehtiin oman tarpeen pohjalta. Työssä luotiin omien vanhojen ohjelmien pohjalta toimiva, monikäyttöinen grafiikkakirjasto ja siihen liittyvä, pääosin englanninkielinen dokumentaatio. Tämän lisäksi tarkoitus oli tehdä myös muutamia yksinkertaisia esimerkkiohjelmia toiminnan esittelemiseksi ja muuntaa vanhempia isompia ohjelmia toimimaan uuden kirjaston päällä suoran OpenGL:n käytön sijaan. Pääkohteena ovat 3D-grafiikkaa käyttävät pelit ja ohjelmat, jotka eivät käytä käyttöjärjestelmien omia käyttöliittymäkomponentteja.

Työssä on käytetty hyväksi ensisijaisesti opiskelun ja työn ohessa opeteltua OpenGL:ää ja samalla hankittu uutta kokemusta kirjasto-tyylisen koodin kirjoittamisesta ja dokumentoinnista.

Suurimmaksi ongelmaksi työn loppupuolella muodostui testaamisen puute. Alun perin tämä johtui rajapintojen jatkuvista muutoksista (mitkä osaltaan johtaisivat jatkuvaan testien päivittämiseen) ja osin myös ajan puutteesta.

Työn lopputuloksena saavutettiin toimiva kirjasto kahdelle hyvin erilaiselle alustalle, sekä alustava versio Linux-mobiilialustoille. Vaikka kirjasto onkin sinänsä toimiva, vaatii se mielestäni muutamia laajempia muutoksia toiminnan yhdenmukaistamiseksi ja laajempaa testaamista muilla käyttäjillä.

Asiasanat:
ohjelmointi, tietokonegrafiikka, ohjelmistot, käyttöliittymä, opengl

ABSTRACT

Oulu University of Applied Sciences
Computer Engineering

Author(s): Juha-Pekka Jokela
Title of thesis: General purpose graphics library
Supervisor(s): Esko Harvala
Term and year of completion: Spring 2013
Number of pages: 18 + appendix

The project was started based on own need. The target was to create a new graphics library based on older projects, and matching english documentation. In addition the goal was to write some simple example programs for demonstrating the functionality, and modifying some older programs to use the library instead of direct OpenGL use.

The work is based on OpenGL standard learned besides studies and work, and new experience regarding library design and documentation that has to be learned during the project.

Towards the end of the project, the biggest problem turned out to be the lack of testing. Originally this was due to unstable API's, which would have resulted in constant need for modifying all relevant test cases. Lack of time was also a major factor.

As a result, I achieved a working library for two very different platforms, and preliminary support for Mobile Linux platforms. Even if the library itself is working, I feel that it needs a few bigger modifications to make the API usage more consistent. Also testing with other users would be needed.

Keywords:
programming, computer graphics, software, user interface, opengl

Sisällys

1 JOHDANTO

Idea työlle lähti omasta tarpeesta. Olen aina harrastanut ohjelmointia vapaa-ajalla, ja lähes kaikissa OpenGL-ohjelmassa olen käyttänyt samaa perustoiminnallisuutta (esimerkiksi ikkunan avaaminen, tietynlaisessa muodossa olevan grafiikan piirto), joten vanhemmissa projekteissa kopioitiin runsaasti koodinpätkiä eri projektien välillä. Ongelmia alkoi tulla viimeistään siinä vaiheessa, kun jotain tiettyä toiminnallisuutta paranneltiin yhdessä projektissa ja se haluttiin ottaa käyttöön muissakin. Useimmiten koodi tuli muokattua sen verran juuri tiettyyn projektiin sopivaksi, että sen muokkaaminen toisessa projektissa toimivaksi vei turhan paljon aikaa ja teki usein koodista yhteensopimatonta edellisen projektin kanssa. Ratkaisuksi tähän aloin siirtää perustoiminnallisuutta omaan projektiin, joka ei saa sisällyttää muiden omien projektien headereita ja jonka tiedostoja kaikki projektit käyttivät suoraan lisäämällä tarvittavat tiedostot Makefileen. Käytännössä tämä pakottaa tekemään koodista sellaista, ettei se suoraan riipu minkään yksittäisen projektin koodista. Itse kutsuisin tätä vaihetta "versio nollaksi".

Luonnollinen jatko tälle on tietenkin tehdä projektista täysin erillinen kirjasto, jota projektit voivat käyttää. Tässä vaiheessa projekteissa käytetty SDL tuntui jo turhalta. Käytännössä sitä käytettiinn vain ikkunan avaamiseen ja eventtien hallintaan, mutta kaikki piirto tapahtui kuitenkin suoraan OpenGL:n kautta. Päätin korvata SDL:n alustakohtaisilla alemman tason kirjastoilla (kuten SDL:kin sisäisesti tekee). Ensimmäisinä olivat GLX (Desktop Linux) ja EGL (Mobiili Linux). SDL:n päätin myös pitää mukana, koska se on paljon laajemmalti tuettu. Mikäli jollekin alustalle ei ole vielä omaa tukea, SDL:n pitäisi toimia.

Kirjaston nimen valinnassa oli käytössä seuraavat kriteerit:

- Samannimistä vastaavaa projektia ei saa olla olemassa.
- Koska projekti käyttää OpenGL:ää, olisi nimessä hyvä olla jossain kohtaa kirjainyhdistelmä ”GL”.
- Nimen pitää olla lyhyt, mahdollisesti akronyymi.

UGLI (Uniform GL Interface) oli pitkän mietinnän jälkeen ensimmäinen kaikki ehdot täyttävä vaihtoehto.

2 MÄÄRITELMÄ

Työn tarkoituksena oli koota aiemmin tehdyissä omissa erillisissä projekteissa olevaa yleiskäyttöiseksi kelpaavaa koodia yhteen, yhtenäistää nimeämiskäytännöt ja rajapinnat, lisätä puuttuvaa toiminnallisuutta ja yhdistää kaikki tämä yleiskäyttöiseksi kirjastoksi. Tarkoituksena oli, että loppukäyttäjä pystyy käyttämään kirjaston headereissa määritettyä toiminnallisuutta, ja kääntämään koodin kirjastoa vastaan. Tarkoituksena oli myös, että kirjaston pystyy tarvittessa vaihtamaan toisin asetuksin käännettyyn versioon ilman, että käyttäjän koodia tarvitsee kääntää uudestaan. Tämä mahdollistaa kirjastoa käyttävien suljetun lähdekoodin ohjelmien julkaisun. Tarkoitus oli myös samanaikaisesti kartoittaa kirjaston ongelmia ja parannusideoita jatkokehitystä varten ja osittain myös toteuttaa niitä työn kuluessa ajan salliessa.

Vaikka tarkoitus on tukea mahdollisimman montaa alustaa, tässä projektissa oli tarkoitus keskittyä lähinnä muutamaan itselle tutumpaan alustaan ja lisätä tukea muille alustoille myöhemmin vaiheittain. On kuitenkin tärkeää huomioida alusta asti, mitkä osat koodista ovat alustariippuvaisia ja mitkä eivät, ja pyrkiä jakamaan alustakohtaiset tiedostot selkeästi omiin hakemistoihinsa.

Itse kirjaston lisäksi tarkoitus oli kirjoittaa myös muutamia pienempiä esimerkkiohjelmia kirjaston testaamista ja toiminnan esittelemistä varten.

Tarkoituksena ei tietenkään ole pitää koodia lopullisesti vain itsellä, vaan ensin julkaista rajapinnat rajalliseen testikäyttöön ja mahdollisesti myöhemmin julkaista koko projekti avoimena lähdekoodina, suhteellisen sallivalla lisenssillä, kuten esimerkiksi LGPL-lisenssillä (GNU Lesser General Public License 2013). Tämä mahdollistaa kirjastoon pohjautuvien ohjelmien tekemisen sekä avoimella tai suljetulla lähdekoodilla.

3 TOIMINTAYMPÄRISTÖ

Työn pääohjelmointikieleksi valitsin C:n, koska itselläni on siitä eniten kokemusta, sekä harrastus- että työpohjalta. Tämän lisäksi tärkeimmät käytetyt rajapinnat (OpenGL, GLX, EGL) on toteutettu C:llä, ja vastaavia C-kirjastoja ei kovin montaa ole (vrt. C++ ja Java). Tulevaisuudessa on tarkoitus tukea myös korkeamman tason kielilä (esimerkiksi Python, Java) ja mahdollisesti myös muita matalamman tason kieliä (esim. C++).

Pääasialliseksi kääntäjäksi valitsin GCC:n (GCC, the GNU Compiler Collection 2013) sen laajan tuen vuoksi. Kääntäjäkohtaisia toimintoa pyrin kuitenkin välttämään silloin, kun se on tarkoituksenmukaista. Kääntäjän virheasetukset määrittelen suhteellisen ankariksi mahdollisten virheiden minimoimiseksi. Vaarattomien virheiden, esimerkiksi käyttämätön muuttuja tai funktio, en anna häiritä kirjaston kääntämistä, mutta pyrin kuitenkin pääsemään näistäkin eroon, kunhan projekti lähestyy loppuaan.

Koska kirjaston on tarkoitus tukea myös mobiililaitteita, rajoittaa tämä olennaisesti sitä, mitä OpenGL:n toiminnallisuutta on käytettävissä. Kirjastoa käännettäessä on päätettävä, käännetäänkö OpenGL ES1 vai OpenGL ES2 versio. Kummatkin versiot toimivat myös laajemmilla OpenGL-toteutuksilla, kunhan vain tarvittava toiminnallisuus löytyy. Tulevaisuudessa on mahdollista lisätä tuki myös muille vastaaville grafiikkakirjastoille.

Koodi on tarkoitus jakaa selkeisiin osiin. Ensisilmäyksellä pitäisi olla selvää, mitkä osat koodista ovat järjestelmäriippuvaisia ja mitkä eivät. Luonnollisesti pyrin myös minimoimaan alustakohtaisen koodin määrän.

Rajapinnat ja niiden käyttö pyritään pitämään mahdollisimman yksinkertaisena. Turhaa toiminnallisuutta en lisää, ennen kuin toiminnallisuutta oikeasti tarvitaan.

Koska tärkeimpänä toiminnallisuutena on grafiikan piirto, on se pyritty toteuttamaan tehokkaimmalla saatavilla olevalla tavalla kuitenkin siten, että piirto toimii myös hitaammalla tavalla silloin, kun nopeampaa tapaa ei ole saatavilla.

Turhaa toiminnallisuutta olen pyrkinyt välttämään. Esimerkiksi OpenGL:n Immediate Modelle ei nykyään ole minkäänlaista järkevää käyttöä (eikä se ole yhteensopiva OpenGL ES1 ja OpenGL ES2 toteutusten kanssa), joten pyrin varmistamaan, että käyttäjä välttäisi tätä ja muuta turhaksi jäänyttä toiminnallisuutta. Tätä varten mm. tarjotaan yleiseen käyttöön paremmin soveltuvia rajapintoja.

Opinnäytetyön tekstin lisäksi luotiin myös englanninkielinen html-pohjainen dokumentaatio kirjaston toiminnasta, jonka on tarkoitus säilyä jatkossa kirjaston virallisena dokumentaationa. Vastaava dokumentaatio löytyy myös työn liittestä 3.

4 TOTEUTUS

4.1 Koodityyli

Koodin sisennykset tehtiin tabulaattorilla. Tarvittavat tasaukset taas tehtiin välilyönnillä.

Muuttujien nimet kirjoitettiin pienellä. Funktioiden ja tyyppien nimet kirjoitettiin käyttäen sekä pieniä että isoja kirjaimia siten, että alkuun laitetaan UGLI -etuliite, ja nimen loppuosassa jokainen sana alkaa isolla alkukirjaimella. UGLI- liitteen jälkeen voitiin myös lisätä valinnainen komponenttiin liittyvä toinen liite, esimerkiksi Layout. Konstruktoreissa ja destruktoreissa New & Delete tulivat heti UGLI etuliitteen jälkeen. Määrittelyt (define), liput ja enumeraatiot kirjoitettiin kokonaan isolla.

4.2 Avoimmuus

Rakenteet pidettiin käyttäjältä suljettuna, paitsi jos niiden avaaminen oli pakollista (esimerkiksi UGLILayoutItem) tai rakenteiden sisältö tulee todennäköisesti pysymään vakiona, ja se mahdollisti hieman nopeamman koodin kirjoittamisen. Yleisesti pyrkimyksenä oli se, että joka tapauksessa vähintään peruskäyttö onnistuu rakenteen sisältöä tuntematta. Joissain tapauksissa (esimerkiksi UGLILayoutItem) käyttäjälle tarpeeton informaatio laitettiin ei-julkiseen rakenteeseen, johon päärakenteesta löytyy osoitin. Tarkoituksena oli pyrkiä mahdollisimman hyvään binääriyhteensopivuuteen. Mikäli kirjaston rakenne käännetään mukaan käyttäjän ohjelmaan, sen sisällön muuttaminen vaatii myös käyttäjän koodin uudelleenkääntämistä. Osoittimilla vastaavaa ongelmaa ei ole.

4.3 Dokumentointi

Dokumentaatio luodaan Doxygenin (Doxygen 2013) avulla. Tämä varmistaa, että rajapinnan headereissa ja erillisessä dokumentaatiossa oleva tieto on keskenään vastaavaa, ja mahdollistaa myös vastaavan dokumentaation luomisen eri muodoissa (html, rtf ym.)

4.4 Peruskomponentit

UGLIMainContext

UGLIMainContextin konstruktori alustaa kirjaston pitäen sisällään koko kirjaston kannalta olennaista informaatiota, joka ei ole sidoksissa esimerkiksi yksittäiseen ikkunaan.

UGLIWindow

UGLIWindow on ikkuna, johon käyttäjä voi piirtää joko kirjaston toiminnallisuutta, tai suoraan OpenGL:ää käyttäen. Myös eventtien hallinta tapahtuu ikkunan kautta. Opinnäytetyössä keskityn lähinnä kirjaston kautta tapahtuvaan piirtoon, mutta suora OpenGL:n kautta tapahtuva piirto on tarkoitus pitää mahdollisena alusta asti, mutta sen toiminnasta ei ole mitään takuita ennen kuin ominaisuutta tuetaan virallisesti jossain myöhemmässä versiossa. Ikkunoita voi olla auki kerrallaan vain yksi, mutta tästä rajoituksesta on tarkoitus hankkiutua eroon tulevaisuudessa. Koska joillain alustoilla ikkuna ei välttämättä sulkeudu kunnolla ohjelman loputtua mikäli käyttäjä ei itse sitä sulje ohjelmallisesti (kun taas esimerkiksi Linuxilla ikkuna sulkeutuu automaattisesti), suljetaan avoinna oleva ikkuna automaattisesti ohjelman loputtua.

UGLIImage

UGLIImage on kuva, joka voi olla joko ohjelmallisesti muodostettu, tai esimerkiksi ladattu tiedostosta. Kuvaa ei voi piirtää suoraan, vaan siitä täytyy ensin tehdä tekstuuri OpenGL:n puolelle. Syynä tälle aluksi turhalta vaikuttavalle välivaiheelle on mahdollisuus ennakkokäsitellä kuvia ennen tekstuuriksi muuntamista. Alustavasti mukana on lähinnä kahden kuvan yhdistäminen pikselikohtaista läpinäkyvyyttä käyttäen, mutta toiminnallisuutta on tarkoitus laajentaa tulevaisuudessa. Myös rajapintaa on tarkoitus parantaa ja yhdenmukaistaa muun kirjaston kanssa.

UGLIPhysics

UGLIPhysics on fysiikan mallinnuksen käsittelyyn liittyvä komponentti, joka ajaa käyttäjän määrittelemät vaihtelevalla ja vakioviiveellä toimivat funktiot edellisestä ajokerrasta olevan viiveen perusteella.

UGLIVectorMath

UGLIVectorMath on kokoelma yleisesti käytettyjä vektorimatematiikkaan liittyviä funktioita. Kirjastossa on erikseen 2- ja 3-ulotteiset versiot.

UGLIMatrixMath

UGLIMatrixMath on kokoelma matriiseihin liittyviä funktioita. Perustoimintojen lisäksi sisältää myös esimerkiksi projektioon liittyvää toiminnallisuutta.

UGLIMatrixStack

UGLIMatrixStack on matriisipino, joka toimii pitkälti samalla tavalla kuin OpenGL:n vastaava. Kirjasto käyttää tätä myös sisäisesti OpenGL ES2 -rajapintaa käytettäessä.

4.5 Käyttöliittymäelementit

Käyttöliittymäelementit ovat yksinkertaisia komponentteja, joita voidaan käyttää käyttöliittymän luomisessa. Nykyisellään valikoima rajoittuu lähinnä informaation välitykseen ohjelmasta käyttäjälle, mutta tätä on tarkoitus laajentaa tulevaisuudessa. Joistakin komponenteista on myös erilliset LayoutItem-versiot.

UGLIUIString

UGLIUIString on merkkijono, jonka sisällön voi muuttaa, ja piirtää näyttöön. Myös numeerisen informaation lisääminen onnistuu pitkälti samalla tavalla kuin printf() -funktiolla.

UGLIUILog

UGLIUILog on loki-ikkuna, joka näyttää ennalta määrätyn määrän viimeksi lisättyjä tekstejä. Tekstiä lisätessä, vanhin teksti poistetaan alhaalta, muita tekstejä siirretään alemmas ja uusi teksti lisätään ylimmälle riville. Komponentista on myös LayoutItem-versio.

UGLIUIFrame

UGLIUIFrame on käyttäjän määrittämästä kuvasta muodostettu kehys, jonka koon voi määrittää vapaasti piirtoa varten. Komponentista ei ole erillistä LayoutItem-versiota, mutta Layout-järjestelmä tukee komponenttia suoraan siten, että jokaiselle LayoutItemille voidaan määrittää omaa kehys.

UGLIUITexture

UGLIUITexture on lähinnä testi- ja esittelykäyttöä varten kehitetty yksinkertainen LayoutItem-komponentti. Konstruktorille annetaan tekstuuri, joka määrittää komponentin minimi- ja maksimikoon. Komponentti piirtää määritetyn tekstuurin komponentille määritettyyn paikkaan.

4.6 Layout-järjestelmä

Layout-järjestelmä koostuu kahdesta peruskomponentista, UGLILayoutItem ja UGLILayoutView. UGLILayoutItem on mikä tahansa ikkunassa oleva komponentti, jonka koon ja sijainnin Layout-järjestelmä määrittää automaattisesti käyttäjän sille määrittelemien minimi- ja maksimiarvojen perusteella. UGLILayoutView taas on komponentti, joka tulee ikkunan ja alimman LayoutItemin väliin. Komponentti on ikkunasta erillinen mikä mahdollistaa piirron ilman Layout-järjestelmää.

Piirron lisäksi järjestelmä osaa myös päätellä, mille komponentille esimerkiksi hiiren klikkauseventit ovat menossa, muuntaa koordinaatit paikallisiksi, ja kutsua oikean LayoutItemin

HandleEvent -funktiota. Myös näppäimistöeventit on mahdollista ohjata sille komponentille, jota käyttäjä on viimeksi hiirellä klikannut.

UGLILayoutView

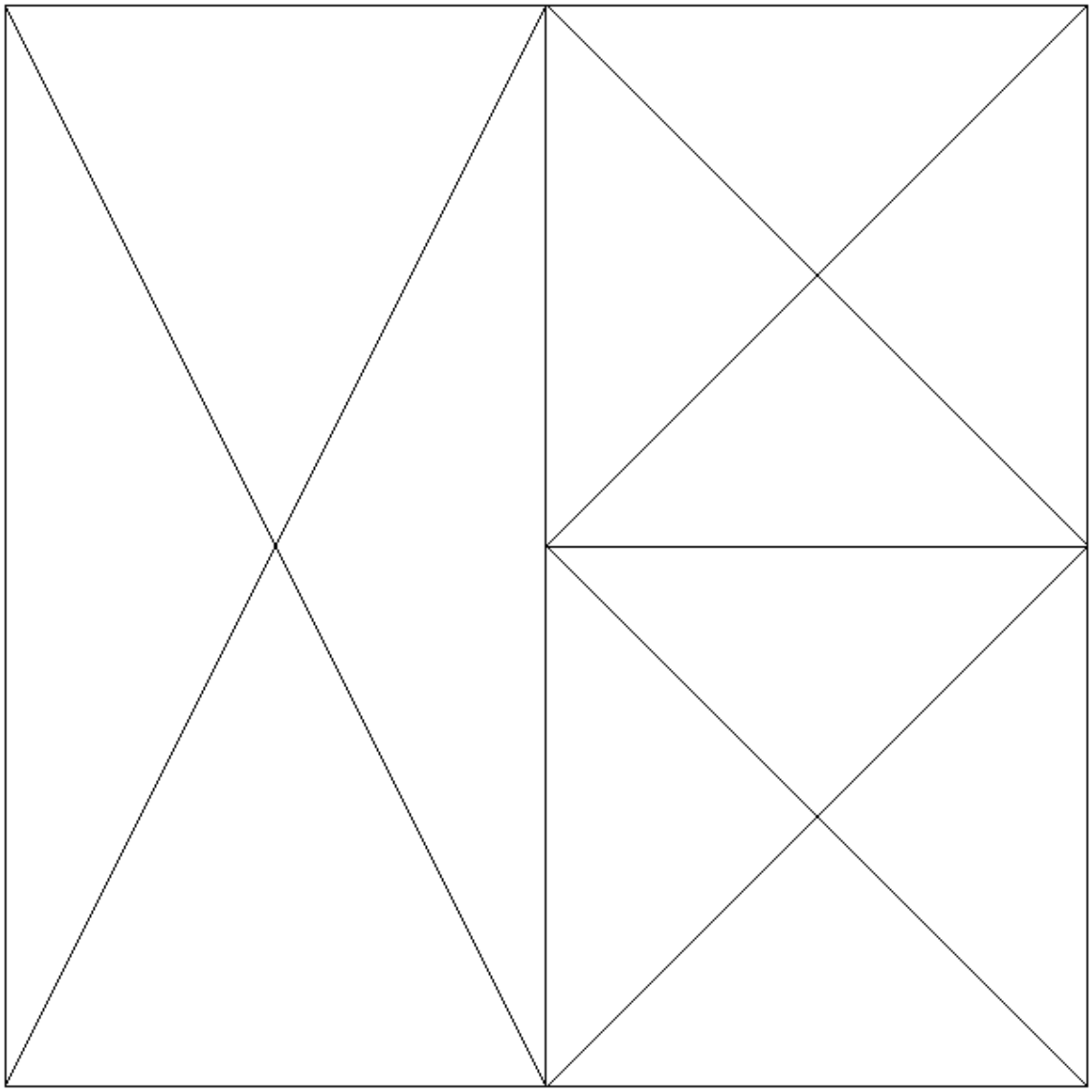
UGLILayoutView on ikkunan ja alimman LayoutItemin välissä oleva komponentti, joka hoitaa eventtien käsittelyn ja edelleenlähettämisen sekä LayoutItemin sijainnin muutokset ikkunan koon muuttuessa.

UGLILayoutItem

UGLILayoutItem on komponentti, jonka pohjalta käyttäjä voi tehdä oman LayoutItemin. Myös kaikki järjestelmän omat layout-komponentit pohjautuvat tähän. Komponentteja voi laittaa myös sisäkkäin, mikäli ulompi komponentti tätä tukee. Sisemmälle komponentille tällä ei ole mitään eroa.

UGLILayoutLinear

UGLILayoutLinear on layout, jossa komponentit järjestetään joko vaaka- tai pystysuoraan riviin. Kirjastosta löytyy myös UGLILayoutHorizontal- ja UGLILayoutVertical-konstruktorit, jotka luovat lineaarisen layoutin nimen määräämään suuntaan.



KUVA 1 *UGLILayoutHorizontal, jonka sisällä UGLILayoutVertical.*

UGLILayoutStack

UGLILayoutStack on layout, jossa komponentit piirretään toistensa päälle. Tämä on hyödyllinen jos halutaan esimerkiksi piirtää ikkuna koko näytön kokoisen näkymän päälle.

5 TESTAUS

Koodin testaaminen rajoitttui lähinnä assert()-kutsuihin ja yksittäisten havaintojen pohjalta tehtyjen ongelmien tarkempaan tutkimiseen. Muistin käyttöön liittyviä ongelmia olen pyrkinyt löytämään ajamalla ohjelmia satunnaisesti Valgrind debuggausohjelman (Valgrind 2013) kanssa, myös silloin kun mitään näkyvää ongelmaa ei ollut.

Erillisten testiohjelmien kirjoittamista lykkäsin osin ajan puutteen takia, ja toisaalta myös sen takia, että kirjaston toiminta on muuttunut olennaisesti sen kehityksen aikana, mikä olisi vaatinut tietenkin aina isompia muutoksia myös testeihin. Matriisioperaatioille kuitenkin kirjoitin testiohjelman, joka vertaa kirjaston omien funktioiden antamia tuloksia OpenGL:n vastaavien tuloksiin. Käytännössä testi on käännettävä ja ajettava erikseen. Mitään hienompia virheraportteja testi ei vielä anna, vaan tuloksia verrataan assert()-funktion avulla, jolloin suoritus pysähtyy ensimmäiseen virheeseen. Jokainen tämän tason virhe on kuitenkin sen verran vakava, että se olisi hyvä korjata välittömästi.

Vaikka kirjoitettu koodi sinällään toimisikin, ongelmaksi muodostui yhdessä tiedostossa GCC-kääntäjän virhe, jonka takia koodista tuli virheellisesti toimivaa. Onnistuin kuitenkin kiertämään virheen lisäämällä muutaman koodirivin, jotka eivät käytännössä tee mitään, mutta jostain syystä korjaavat funktion toiminnan kirjoitettua koodia vastaavaksi.

Mitään tämän parempaa testausta ei toistaiseksi ole, ja se on alkanut vähitellen muodostua ongelmaksi samalla, kun kirjaston toiminta on alkanut vakaantua ja testien kirjoittaminen alkaa tuntua mielekkäämmältä.

6 JATKOKEHITYSMAHDOLLISUUDET

Työn aikana tuli vastaan tiettyjä ongelmia, joita on tarkoitus tutkia paremmin tulevaisuudessa. Ehkä tärkein on OpenGL ES2-tuen korjaaminen. Alustava, osin puutteellinen tuki löytyy, mutta lopullinen toteutus vaatii osittaisen uudelleenkirjoittamisen. Tämän lisäksi on useita komponentteja, joiden toiminnan korjaamisen ja viimeistelyn olen suosiolla jättänyt myöhemmäksi. Toistaiseksi ne on poistettu kirjastosta.

Myös tuettuja alustoja on tarkoitus laajentaa. Parhaiten tuettu ikkunanhallintajärjestelmä on tällä hetkellä X11 GLX:llä. Tämän lisäksi myös SDL ja MorphOS ovat tuettuja, ja EGL-versio toimii osittain. Nykyisten ongelmien korjaamisen jälkeen tarkoituksena on lisätä tuki kokonaan uusille alustoille. Näistä ensimmäisinä tulevat mieleen MacOS X ja Applen mobiililaitteet.

Testausta on tarkoitus laajentaa lisäämällä uusia testiohjelmia, jotka varmistavat kirjaston toiminnallisuutta varsinkin niiltä osin, kun samaa toiminnallisuutta toteutetaan usealla eri rajapinnalla.

Kirjastoa voisi laajentaa useampaankin eri suuntaan. Ensimmäisinä tulevat mieleen mm. ääni- ja verkkotuki. Toki näiden käyttö onnistuu myös erillisillä kirjastoilla, joten suurin kysymys on, tarvitaanko näitä tällaisessa kirjastossa vai ei. Koodiin on tehty alustava rajapinta 3D-näytölle piirtämiseen. Käytännössä tämä toistaiseksi antaa vain mahdollisuuden kertoa käyttäjän komponentille, kumman silmän kuva milloinkin piirretään. Nykyisellään näin ei kuitenkaan koskaan tehdä, mutta parametrit on lisätty, koska niitä on tarkoitus käyttää tulevaisuudessa. Käytännön toteutus tulee vaatimaan tarkempaa perehtymistä mm. siihen, miten tätä mahdollisesti käytetään kirjaston sisäisten komponenttien piirtämisessä. Mahdollisesti käyttäjä voisi esimerkiksi antaa päällekäin oleville komponenteille eri syvyysarvot. Käytännössä tämä vaikuttaa tietenkin myös hiirieventteihin.

Layout-järjestelmän alakomponenttien lisääminen ei tällä hetkellä onnistu kahden olemassa olevan komponentin väliin, ellei siihen ole ennalta tarkoituksella jätetty tyhjää väliä. Nykyinen järjestelmä kuitenkin toimii hyvin sellaisissa tilanteissa, jossa ikkunan komponenttien asettelu on ennalta tunnettu.

7 YHTEENVETO

Työn tuloksena saatiin toimiva runko kirjastolle, joka tosin vaatii vielä hienosäätöä ennen laajempaa julkaisua. Kovin montaa alustaa ei vielä tässä vaiheessa ole tuettuna, mutta koodista on selkeästi pystytty eristämään alustakohtaiset osat, mikä osaltaan helpottaa tuen laajentamista jatkossa. Alustava tuki löytyy sekä Ubuntu Linuxille (Ubuntu 2013) että MorphOS:lle (MorphOS 2013), jotka ovat hyvinkin erilaisia käyttöjärjestelmiä, mikä mielestäni todistaa kirjaston yleiskäyttöisyyden erilaisilla alustoilla.

Kaikista osa-alueista ehkä testaus jäi puutteellisimmaksi. Toisaalta taas laajempi testaus olisi vienyt aikaa muulta kehitykseltä.

Rajapintojen dokumentointi osoittautui myös hieman luultua hankalammaksi. HTML-version kanssa ei ollut juurikaan ongelmia (ja tämä on juuri se versio, mitä loppukäyttäjät todennäköisesti eniten käyttävät), mutta RTF-muotoisen version kanssa tuli ongelmia mm. formatoinnissa. Koska kuitenkin tärkeimmän muodon kanssa ei ollut suurempia ongelmia, tulen varmaan jatkossakin käyttämään samaa ohjelmaa dokumentaation hallintaan.

LÄHDELUETTELO

Doxygen Haettu 19. 3. 2013. http://doxygen.org/

GCC, the GNU Compiler Collection Haettu 19. 3. 2013. http://gcc.gnu.org/

GNU Lesser General Public License Haettu 19.3.2013. http://www.gnu.org/copyleft/lesser.html

MorphOS Haettu 19. 3. 2013. http://morphos-team.net/

OpenGL Haettu 19. 3. 2013. http://www.opengl.org/

SDL Haettu 19. 3. 2013. http://www.libsdl.org/

Ubuntu Linux Haettu 19. 3. 2013. http://ubuntu.com/

Valgrind Haettu 19. 3. 2013. http://valgrind.org/

LIITTEET

Liite 1: Koodi, joka avaa ikkunan, joka koostuu yhdestä LayoutItemistä

```
#include <ugli/window.h>

#include <ugli/texture.h>

#include <ugli/layout/layoutitem.h>

#include <ugli/layout/layoutview.h>

#include <ugli/uielements/textureli.h>

int main(int argc,char *argv[])

{

        UGLIMainContext context=UGLINewMainContext(argc, argv);

        UGLIWindow window=UGLIWindowOpen(context, UGLI_T_WINDOW_DONE);

        UGLILayoutItem   li=UHLIUINewTextureLI(UGLINewTextureFromFile(”hello.png”,
        UGLI_F_TEXTURE_NONE));

        UGLILayoutView lv=UGLINewLayoutView(li, NULL);

        UGLILayoutViewRunEventLoop(lv);

        return 0;

}
```

Liite 2: Koodi, joka toteuttaa yksinkertaisen LayoutItemin

```
#include <ugli/memory.h>
#include <ugli/matrixmathinit.h>
#include <ugli/vertexarray.h>
#include <ugli/vertexsizes.h>
#include <ugli/glstate.h>
#include <ugli/glcolor.h>
#include <ugli/layout/layoutitem.h>
#include <ugli/layout/layoutitem_struct.h>
#define NUM_VERTICES 4
#define FLOATS_VERTEX (UGLI_FLOATS_V)

typedef struct SimpleLIData *SimpleLI;

struct SimpleLIData
{
        UGLIVertexArray va;
        float color[UGLI_FLOATS_C];
};
```

```
static void MyDelete(UGLILayoutItem li)
{
        if(li)
        {
                SimpleLI sli=(SimpleLI)UGLILayoutItemGetData(li);
                UGLIDeleteVertexArray(sli->va);
        }
}

static UGLILayoutItemDrawStatusFlag MyDraw(const UGLILayoutItem li, UGLITicks delta, UGLILayoutItemDrawFlag drawflags)
{
        SimpleLI sli=(SimpleLI)UGLILayoutItemGetData(li);
        UGLIDisable(UGLI_DEPTH_TEST);
        UGLIEnable(UGLI_BLEND);
        UGLIColor4fv(sli->color);
        UGLIVertexArrayDraw(sli->va);
        return UGLI_LI_F_DRAW_COLOR_BUFFER;
}
```

```
UGLILayoutItem NewSimpleLI(const float *color)
{
        const float va[]={
        -1.0f,-1.0f, 0.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 1.0f,
        1.0f,-1.0f, 0.0f, 1.0f,
        -1.0f, 1.0f, 0.0f, 1.0f};
        SimpleLI sli=UGLIMalloc(sizeof(struct SimpleLIData));
        UGLILayoutItem li=UGLINewLayoutItem(&MyDelete, &MyDraw, NULL, NULL, NULL, NULL, NULL, UGLI_LI_CLEAR_COLOR, (void*)sli, UGLI_LI_F_SET_VIEWPORT | UGLI_LI_F_CREATE_PROJECTION);
        sli->va=UGLINewVertexArray(UGLI_LINES, UGLI_VA_STATIC_VERTICES | UGLI_VA_STATIC_LENGTH);
        UGLIVertexArraySetArrays(sli->va, va, NUM_VERTICES, NULL, 0);
        UGLIVertexArrayFinish(sli->va);
        memcpy(sli->color, color, sizeof(float[UGLI_FLOATS_C]));
        return li;
}
```

Data Structure Documentation

UGLIEventData Union Reference

Union consisting of all event types.

`#include <events.h>`

Data Fields

UGLIEventType type
UGLIWhichLayoutItemEvent whichlayoutevent
UGLIMouseButtonEvent mousebuttonevent
UGLIMouseMoveEvent mousemoveevent
UGLIKeyboardEvent keyboardevent
UGLIRedrawRequestEvent redrawrequestevent
UGLISetMouseMotionAcceptEvent setmousemotionacceptevent
UGLIUserEvent userevent

Detailed Description

Union consisting of all event types.

Field Documentation

UGLIKeyboardEvent UGLIEventData::keyboardevent

Keyboard event.

UGLIMouseButtonEvent UGLIEventData::mousebuttonevent

Mouse button event.

UGLIMouseMoveEvent UGLIEventData::mousemoveevent

Mouse move event.

UGLIRedrawRequestEvent UGLIEventData::redrawrequestevent

Redraw request event.

UGLISetMouseMotionAcceptEvent UGLIEventData::setmousemotionacceptevent

Mouse motion accept event.

UGLIEventType UGLIEventData::type

Event type.

UGLIUserEvent UGLIEventData::userevent

User event.

UGLIWhichLayoutItemEvent UGLIEventData::whichlayoutevent

WhichLayoutItem event.

The documentation for this union was generated from the following file:

1 **events.h**

UGLIIntrusiveLinkedList Struct Reference

`#include <intrusivelinkedlist.h>`

Data Fields

UGLIIntrusiveLinkedListElement * first
UGLIIntrusiveLinkedListElement * last

Detailed Description

Intrusive linked list struct.

Field Documentation

UGLIIntrusiveLinkedListElement* UGLIIntrusiveLinkedList::first

Link to the first IntrusiveLinkedListElement (or NULL if none).

UGLIIntrusiveLinkedListElement* UGLIIntrusiveLinkedList::last

Link to the last IntrusiveLinkedListElement (or NULL if none).

The documentation for this struct was generated from the following file:

2 intrusivelinkedlist.h

UGLIIntrusiveLinkedListElement Struct Reference

```
#include <intrusivelinkedlist.h>
```

Data Fields

struct
UGLIIntrusiveLinkedListElement * next
struct
UGLIIntrusiveLinkedListElement * previous
struct **UGLIIntrusiveLinkedList * list**

Detailed Description

Intrusive linked list element struct.

Field Documentation

struct UGLIIntrusiveLinkedList* UGLIIntrusiveLinkedListElement::list

Link to next IntrusiveLinkedList the element belongs to.

struct UGLIIntrusiveLinkedListElement* UGLIIntrusiveLinkedListElement::next

Link to next IntrusiveLinkedListElement (or NULL if none).

struct UGLIIntrusiveLinkedListElement* UGLIIntrusiveLinkedListElement::previous

Link to previous IntrusiveLinkedListElement (or NULL if none).

The documentation for this struct was generated from the following file:

3 intrusivelinkedlist.h

UGLIKeyboardEvent Struct Reference

Keyboard event.

`#include <events.h>`

Data Fields

UGLIEventType type
unsigned int **key**
int **state**

Detailed Description

Keyboard event.

Field Documentation

unsigned int UGLIKeyboardEvent::key

Key.

int UGLIKeyboardEvent::state

Key state.

UGLIEventType UGLIKeyboardEvent::type

Event type.

The documentation for this struct was generated from the following file:

4 events.h

UGLILayoutItemData Struct Reference

LayoutItem struct for code that implements new UGLILayoutItem.

`#include <layoutitem_struct.h>`

Data Fields

UGLILayoutItemFlag flags
UGLILayoutItem parent
UGLILayoutItemPrivate p
UGLILayoutItemHidden hidden
unsigned int **hidepriority**
int **minmargin** [2]
int **maxmargin** [2]
int **totalmargin** [2]
int **minsize** [2]
int **maxsize** [2]
int **prefsize** [2]
float **weight** [2]
int **contentpos** [2]
int **contentsize** [2]
UGLIEventType acceptedevents
float * **projectionmatrix**

Detailed Description

LayoutItem struct for code that implements new UGLILayoutItem.

Contains all items the user might need to access.
All items the user doesn't need to know are stored inside the private struct.

Field Documentation

UGLIEventType UGLILayoutItemData::acceptedevents

Which kinds of events the LayoutItem accepts.

int UGLILayoutItemData::contentpos[2]

X & Y position of contents.

int UGLILayoutItemData::contentsize[2]

width & height of contents.

UGLILayoutItemFlag UGLILayoutItemData::flags

Flags.

UGLILayoutItemHidden UGLILayoutItemData::hidden

If any of the UGLILayoutItemHidden flags are high, the UGLILayoutItem is hidden.

unsigned int UGLILayoutItemData::hidepriority

Defines at which order LayoutItems will be hidden due to lack of space. UNUSED

int UGLILayoutItemData::maxmargin[2]

Max (left & bottom) margins. Doesn't include frame.

int UGLILayoutItemData::maxsize[2]

Maximum size the LayoutItem will be drawn at. MAXSIZE_NO_LIMIT for no limit.

int UGLILayoutItemData::minmargin[2]

Min (right & top) margins. Doesn't include frame.

int UGLILayoutItemData::minsize[2]

Minimum size the LayoutItem will be drawn at.

UGLILayoutItemPrivate UGLILayoutItemData::p

Private data.

UGLILayoutItem UGLILayoutItemData::parent

Parent LayoutItem (or NULL for none).

int UGLILayoutItemData::prefsize[2]

Preferred size. No LayoutItem should get extra space before all in layout have reached this. Currently unused.

float* UGLILayoutItemData::projectionmatrix

Projection matrix to set before Draw() is called (or NULL for none).

int UGLILayoutItemData::totalmargin[2]

Total X & Y margins, including frame.

float UGLILayoutItemData::weight[2]

X & Y weight of contents (how big part of extra space the LayoutItem will get).

The documentation for this struct was generated from the following file:

5 layout/**layoutitem_struct.h**

UGLIMouseButtonEvent Struct Reference

Mouse button event.

`#include <events.h>`

Data Fields

UGLIEventType type
int **xpos**
int **ypos**
int **button**
int **state**

Detailed Description

Mouse button event.

Field Documentation

int UGLIMouseButtonEvent::button

Mouse button.

int UGLIMouseButtonEvent::state

Mouse button state.

UGLIEventType UGLIMouseButtonEvent::type

Event type.

int UGLIMouseButtonEvent::xpos

X Position.

int UGLIMouseButtonEvent::ypos

Y Position.

The documentation for this struct was generated from the following file:

6 events.h

UGLIMouseMoveEvent Struct Reference

Mouse move event.

`#include <events.h>`

Data Fields

UGLIEventType type
int **xpos**
int **ypos**

Detailed Description

Mouse move event.

Field Documentation

UGLIEventType UGLIMouseMoveEvent::type

Event type.

int UGLIMouseMoveEvent::xpos

X Position.

int UGLIMouseMoveEvent::ypos

Y Position.

The documentation for this struct was generated from the following file:

7 events.h

UGLIPointerArray Struct Reference

Pointer array.

`#include <pointerarray.h>`

Data Fields

void ** **array**
volatile unsigned int **size**
unsigned int **maxsize**

Detailed Description

Pointer array.

Field Documentation

void UGLIPointerArray::array**

Array for pointers.

unsigned int UGLIPointerArray::maxsize

(Current) max. size.

volatile unsigned int UGLIPointerArray::size

Current size (max. number of pointers).

The documentation for this struct was generated from the following file:

8 pointerarray.h

UGLIQuat Struct Reference

`#include <quat.h>`

Data Fields

float **x**
float **y**
float **z**
float **w**

Detailed Description

Quaternion object.

Field Documentation

float UGLIQuat::w

W component of quaternion.

float UGLIQuat::x

X component of quaternion.

float UGLIQuat::y

Y component of quaternion.

float UGLIQuat::z

Z component of quaternion.

The documentation for this struct was generated from the following file:

9 quat.h

UGLIReallocArray Struct Reference

Realloc array.

`#include <reallocarray.h>`

Data Fields

void * **array**
volatile size_t **size**
size_t **maxsize**
size_t **elementsize**

Detailed Description

Realloc array.

ReallocTable is a container, where all elements are guaranteed to be in the same array, as required by f.ex. vertex arrays.

Field Documentation

void* UGLIReallocArray::array

Array for elements.

size_t UGLIReallocArray::elementsize

Size of a single element (in bytes).

size_t UGLIReallocArray::maxsize

(Current) array maximum size.

volatile size_t UGLIReallocArray::size

Array size (how many elements).

The documentation for this struct was generated from the following file:

10 reallocarray.h

UGLIRedrawRequestEvent Struct Reference

Redraw request event.

`#include <events.h>`

Data Fields

UGLIEventType type

Detailed Description

Redraw request event.

Field Documentation

UGLIEventType UGLIRedrawRequestEvent::type

Event type.

The documentation for this struct was generated from the following file:

11 events.h

UGLISetMouseMotionAcceptEvent Struct Reference

Mouse motion accept event.
`#include <events.h>`

Data Fields

UGLIEventType type
int **accept**

Detailed Description

Mouse motion accept event.

Field Documentation

int UGLISetMouseMotionAcceptEvent::accept

Whether or not mouse motion events are accepted.

UGLIEventType UGLISetMouseMotionAcceptEvent::type

Event type.

The documentation for this struct was generated from the following file:

12 events.h

UGLIUserEvent Struct Reference

User defined event.
`#include <events.h>`

Data Fields

UGLIEventType type
UserEventType usereventtype
void * **data1**
void * **data2**

Detailed Description

User defined event.

Field Documentation

void* UGLIUserEvent::data1

Data pointer 1.

void* UGLIUserEvent::data2

Data pointer 2.

UGLIEventType UGLIUserEvent::type

Event type.

UserEventType UGLIUserEvent::usereventtype

User-defined event ID.

The documentation for this struct was generated from the following file:

13 events.h

UGLIWhichLayoutItemEvent Struct Reference

WhichLayoutItem event.

`#include <events.h>`

Data Fields

UGLIEventType type
int **xpos**
int **ypos**

Detailed Description

WhichLayoutItem event.

Field Documentation

UGLIEventType UGLIWhichLayoutItemEvent::type

Event type.

int UGLIWhichLayoutItemEvent::xpos

X Position.

int UGLIWhichLayoutItemEvent::ypos

Y Position.

The documentation for this struct was generated from the following file:

14 events.h

File Documentation

angle.h File Reference

Angle conversion between radians and degrees.

```
#include <math.h>
#include "header_begin.h"
#include "header_end.h"
```

Detailed Description

Angle conversion between radians and degrees.

---

average.h File Reference

Average value objects.

```
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct UGLIAverageFData * **UGLIAverageF**
typedef struct UGLIAverageIData * **UGLIAverageI**

Functions

UGLIAverageF NewAverageF (unsigned int values, float initialvalue)
UGLIAverageI NewAverageI (unsigned int values, int initialvalue)
void **DeleteAverageF** (**UGLIAverageF** average)
void **DeleteAverageI** (**UGLIAverageI** average)
float **UpdateAverageF** (**UGLIAverageF** average, float value)
int **UpdateAverageI** (**UGLIAverageI** average, int value)

Detailed Description

Average value objects.

Functions calculating average of last n entries.

Typedef Documentation

typedef struct UGLIAverageFData* UGLIAverageF

AverageF object.

typedef struct UGLIAverageIData* UGLIAverageI

AverageI object.

Function Documentation

void DeleteAverageF (UGLIAverageF *average*)

Destructor that deletes the specified AverageF object.

Parameters:

average	UGLIAverageF object to delete.

void DeleteAverageI (UGLIAverageI *average*)

Destructor that deletes the specified AverageI object.

Parameters:

average	UGLIAverageI object to delete.

UGLIAverageF NewAverageF (unsigned int *values*, float *initialvalue*)

Constructor that creates a new UGLIAverageF object.

Parameters:

values	How many last values to get the average from.
initialvalue	Initial value for all entries.

new UGLIAverageF object (or NULL if not successful).

UGLIAverageI NewAverageI (unsigned int *values*, int *initialvalue*)

Constructor that creates a new UGLIAverageI object.

Parameters:

Returns:

values	How many last values to get the average from.
initialvalue	Initial value for all entries.

new UGLIAverageI object (or NULL if not successful).

float UpdateAverageF (UGLIAverageF *average*, float *value*)

Appends new value & returns new average.

Parameters:

Returns:

average	UGLIAverageF object to add new value to.
value	Value to add.

new average value.

int UpdateAverageI (UGLIAverageI *average*, int *value*)

Appends new value & returns new average.

Parameters:

Returns:

average	UGLIAverageI object to add new value to.
value	Value to add.

new average value.

---

condvar.h File Reference

Condition variable (for multithreading).

```
#include "mutex.h"
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct UGLICondVarData * **UGLICondVar**

Functions

UGLICondVar UGLINewCondVar (void)
void **UGLIDeleteCondVar** (**UGLICondVar** cv)
void **UGLICondWait** (**UGLICondVar** cv, **UGLIMutex** mutex)
void **UGLICondSignal** (**UGLICondVar** cv)
void **UGLICondBroadcast** (**UGLICondVar** cv)

Detailed Description

Condition variable (for multithreading).

Typedef Documentation

typedef struct UGLICondVarData* UGLICondVar

Condition variables enable one or more threads to wait for some event.

Function Documentation

void UGLICondBroadcast (UGLICondVar *cv*)

Signals all threads waiting for the specified condition variable. The threads will return from GenCondWait().

Parameters:

Returns:

cv	condition variable to signal.

void UGLICondSignal (UGLICondVar *cv*)

Signals the thread that is waiting for the specified CondVar. This will cause the mutex specified for CondWait to be locked, and the waiting thread to be activated. Signals at least one thread waiting for the specified condition variable. The thread will return from GenCondWait().

Parameters:

cv	condition variable to signal.

void UGLICondWait (UGLICondVar *cv*, UGLIMutex *mutex*)

mutex should be locked when calling. Will block the current thread on specified CondVar. Will unlock the mutex on call.

Will re-lock the mutex when the specified CondVar is signaled. Makes current thread wait for the specified condition variable.

Parameters:

cv	Condition variable to wait for.
mutex	Locked mutex (protecting the condition variable). Will be unlocked after call.

void UGLIDeleteCondVar (UGLICondVar *cv*)

Deletes the specified conditional variable.

Parameters:

cv	condition variable to delete.

UGLICondVar UGLINewCondVar (void)

Creates new condition variable.

Returns:

new condition variable (or NULL if not successful).

---

events.h File Reference

Event handling.

```
#include "header_begin.h"
#include "header_end.h"
```

Data Structures

struct **UGLIWhichLayoutItemEvent**
WhichLayoutItem event. struct **UGLIMouseButtonEvent**
Mouse button event. struct **UGLIMouseMoveEvent**
Mouse move event. struct **UGLIKeyboardEvent**
Keyboard event. struct **UGLIRedrawRequestEvent**
Redraw request event. struct **UGLISetMouseMotionAcceptEvent**
Mouse motion accept event. struct **UGLIUserEvent**
User defined event. union **UGLIEventData**
Union consisting of all event types. **Typedefs**

typedef unsigned int **UserEventType**
typedef union **UGLIEventData * UGLIEventPointer**
typedef union **UGLIEventData UGLIEvent**
Union consisting of all event types.

Enumerations

enum **UGLIEventHandleFlag** { **UGLIEventHandle_None** = 0x0000, **UGLIEventHandle_EventAccepted** = 0x0001, **UGLIEventHandle_NeedsRedraw** = 0x0002, **UGLIEventHandle_ProgramQuit** = 0x0004 }
enum **UGLIKeySymbol** { **UGLI_KEY_UNDEFINED**, **UGLI_KEY_0**, **UGLI_KEY_1**, **UGLI_KEY_2**, **UGLI_KEY_3**, **UGLI_KEY_4**, **UGLI_KEY_5**, **UGLI_KEY_6**, **UGLI_KEY_7**, **UGLI_KEY_8**, **UGLI_KEY_9**, **UGLI_KEY_a**, **UGLI_KEY_b**, **UGLI_KEY_c**, **UGLI_KEY_d**, **UGLI_KEY_e**, **UGLI_KEY_f**,

UGLI_KEY_g, **UGLI_KEY_h**, **UGLI_KEY_i**, **UGLI_KEY_j**, **UGLI_KEY_k**, **UGLI_KEY_l**, **UGLI_KEY_m**, **UGLI_KEY_n**, **UGLI_KEY_o**, **UGLI_KEY_p**, **UGLI_KEY_q**, **UGLI_KEY_r**, **UGLI_KEY_s**, **UGLI_KEY_t**, **UGLI_KEY_u**, **UGLI_KEY_v**, **UGLI_KEY_w**, **UGLI_KEY_x**, **UGLI_KEY_y**, **UGLI_KEY_z**, **UGLI_KEY_SPACE**, **UGLI_KEY_ENTER**, **UGLI_KEY_UP**, **UGLI_KEY_DOWN**, **UGLI_KEY_LEFT**, **UGLI_KEY_RIGHT**, **UGLI_KEY_LSHIFT**, **UGLI_KEY_RSHIFT**, **UGLI_KEY_LCONTROL**, **UGLI_KEY_RCONTROL**, **UGLI_KEY_LALT**, **UGLI_KEY_RALT** }

enum **UGLIEventType** { **UGLIEventUndefined** = 0, **UGLIEventWhichLayoutItem**, **UGLIEventMouseButton**, **UGLIEventMouseMove**, **UGLIEventKeyboard**, **UGLIEventUserEvent**, **UGLIEventRequestRedraw**, **UGLIEventSetMouseMotionAccept** }

enum **UGLIMouseButtonId** { **UGLI_MOUSEBUTTON_LEFT** = 1, **UGLI_MOUSEBUTTON_MIDDLE** = 2, **UGLI_MOUSEBUTTON_RIGHT** = 3 }

Mouse buttons. enum **UGLIMouseButtonState** { **UGLI_MOUSEBUTTON_UP**, **UGLI_MOUSEBUTTON_DOWN** }

Mouse button states. **Functions**

void **UGLIMouseGetPos** (int *x, int *y)

Detailed Description

Event handling.

Event system supports following types of events

UGLIWhichLayoutItemEvent
UGLIMouseButtonEvent
UGLIMouseMoveEvent
UGLIKeyboardEvent
UGLIRedrawRequestEvent
UGLISetMouseMotionAcceptEvent
UGLIUserEvent

Typedef Documentation

typedef union UGLIEventData* UGLIEventPointer

Pointer to event.

typedef unsigned int UserEventType

User defined event type. All values are free to use.

Enumeration Type Documentation

enum UGLIEventHandleFlag

Flags returned from event handlers.

Enumerator:

UGLIEventHandle_None No flags.
UGLIEventHandle_EventAccepted Event accepted, shouldn't be sent to further objects.
UGLIEventHandle_NeedsRedraw Program wants immediate redraw.
UGLIEventHandle_ProgramQuit Program wants to quit.

enum UGLIEventType

Supported event types.

Enumerator:

UGLIEventUndefined undefined / unknown event type.
UGLIEventWhichLayoutItem WhichLayoutItem event.
UGLIEventMouseButton Mouse button event.
UGLIEventMouseMove Mouse move event.
UGLIEventKeyboard Keyboard event.
UGLIEventUserEvent User event.
UGLIEventRequestRedraw Redraw request event.
UGLIEventSetMouseMotionAccept Mouse motion accept event.

enum UGLIKeySymbol

List of key symbols.

Enumerator:

UGLI_KEY_UNDEFINED Currently undefined key symbol.
UGLI_KEY_0 0 key.
UGLI_KEY_1 1 key.
UGLI_KEY_2 2 key.
UGLI_KEY_3 3 key.
UGLI_KEY_4 4 key.
UGLI_KEY_5 5 key.
UGLI_KEY_6 6 key.
UGLI_KEY_7 7 key.
UGLI_KEY_8 8 key.
UGLI_KEY_9 9 key.
UGLI_KEY_a a key.
UGLI_KEY_b b key.
UGLI_KEY_c c key.
UGLI_KEY_d d key.
UGLI_KEY_e e key.
UGLI_KEY_f f key.
UGLI_KEY_g g key.
UGLI_KEY_h h key.
UGLI_KEY_i i key.
UGLI_KEY_j j key.

UGLI_KEY_k k key.
UGLI_KEY_l l key.
UGLI_KEY_m m key.
UGLI_KEY_n n key.
UGLI_KEY_o o key.
UGLI_KEY_p p key.
UGLI_KEY_q q key.
UGLI_KEY_r r key.
UGLI_KEY_s s key.
UGLI_KEY_t t key.
UGLI_KEY_u u key.
UGLI_KEY_v v key.
UGLI_KEY_w w key.
UGLI_KEY_x x key.
UGLI_KEY_y y key.
UGLI_KEY_z z key.
UGLI_KEY_SPACE space key.
UGLI_KEY_ENTER enter key. UNUSED.
UGLI_KEY_UP cursor up key.
UGLI_KEY_DOWN cursor down key.
UGLI_KEY_LEFT cursor left key.
UGLI_KEY_RIGHT cursor right key.
UGLI_KEY_LSHIFT left shift key.
UGLI_KEY_RSHIFT right shift key.
UGLI_KEY_LCONTROL left control key.
UGLI_KEY_RCONTROL right control key.
UGLI_KEY_LALT left alt key.
UGLI_KEY_RALT right alt key.

enum UGLIMouseButtonId

Mouse buttons.

Enumerator:

UGLI_MOUSEBUTTON_LEFT Left mouse button.
UGLI_MOUSEBUTTON_MIDDLE Middle mouse button.
UGLI_MOUSEBUTTON_RIGHT Right mouse button.

enum UGLIMouseButtonState

Mouse button states.

Enumerator:

UGLI_MOUSEBUTTON_UP Mouse button up.
UGLI_MOUSEBUTTON_DOWN Mouse button down.

Function Documentation

void UGLIMouseGetPos (int * *x*, int * *y*)

Returns current mouse position within the window.

Parameters:

x	Space for X position.
y	Space for Y position.

---

glcolor.h File Reference

Color.

```
#include "header_begin.h"
#include "header_end.h"
```

Functions

void **UGLIColor4f** (float r, float g, float b, float a)
void **UGLIColor3f** (float r, float g, float b)
void **UGLIColor4fv** (const float *col)
void **UGLIColor3fv** (const float *col)
void **UGLIColorDefault** (void)
void **UGLIColorPrevious** (void)

Detailed Description

Color.

Function Documentation

void UGLIColor3f (float *r*, float *g*, float *b*)

Sets RGB color.

Parameters:

r	Red component.
g	Green component.
b	Blue component.

void UGLIColor3fv (const float * *col*)

Sets RGB color.

Parameters:

col	Pointer to array with all 3 components.

void UGLIColor4f (float *r*, float *g*, float *b*, float *a*)

Sets RGBA color.

Parameters:

r	Red component.
g	Green component.
b	Blue component.
a	Alpha component.

void UGLIColor4fv (const float * *col*)

Sets RGBA color.

Parameters:

col	Pointer to array with all 4 components.

void UGLIColorDefault (void)

Sets default color (white).

void UGLIColorPrevious (void)

Sets previous color (after specifying colors in vertex array).

---

glfont.h File Reference

Font handling.

```
#include "texture.h"
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct UGLIFontData * **UGLIFont**

Functions

UGLIFont UGLINewFont (const char *filename, unsigned int charsx, unsigned int charsy)
void **UGLIDeleteFont** (**UGLIFont** font)
void **UGLIFontSetCurrent** (const **UGLIFont** font)
void **UGLIFontReset** (void)

Detailed Description

Font handling.

Typedef Documentation

typedef struct UGLIFontData* UGLIFont

UGLIFont object.

Function Documentation

void UGLIDeleteFont (UGLIFont *font*)

Destructor that deletes the specified UGLIFont object.

Parameters:

font	UGLIFont to delete.

void UGLIFontReset (void)

"Unselect" current font. Should be called after the texture might have changed since last text drawing.

void UGLIFontSetCurrent (const UGLIFont *font*)

Select specified font.

Parameters:

font	UGLIFont to select.

UGLIFont UGLINewFont (const char * *filename*, unsigned int *charsx*, unsigned int *charsy*)

Constructor that creates a new UGLIFont object.

Parameters:

filename	File to load font graphics from.
charsx	How many characters does the texture have in X dimension.
charsy	How many characters does the texture have in Y dimension.

---

glprimitive.h File Reference

Primitives that can be drawn.

```
#include "header_begin.h"
```

```
#include "header_end.h"
```

Enumerations

enum **UGLIPrimitive** { **UGLI_POINTS**, **UGLI_LINES**, **UGLI_LINE_STRIP**, **UGLI_TRIANGLES**, **UGLI_TRIANGLE_STRIP**, **UGLI_TRIANGLE_FAN** }

Detailed Description

Primitives that can be drawn.

Enumeration Type Documentation

enum UGLIPrimitive

Primitives that user can draw.

Enumerator:

UGLI_POINTS Points, one vertex each.
UGLI_LINES Lines, two vertices each.
UGLI_LINE_STRIP Line strip, next line begins from the end vertex of the previous line.
UGLI_TRIANGLES Triangles, three vertices each.
UGLI_TRIANGLE_STRIP Triangle strip.
UGLI_TRIANGLE_FAN Triangle fan, each triangle starts from the first vertex, and shares the previous one.

---

glstate.h File Reference

Functionality to enable / disable various features.

```
#include "header_begin.h"
#include "header_end.h"
```

Enumerations

enum **UGLIFeature** { **UGLI_DEPTH_TEST**, **UGLI_BLEND**, **UGLI_LIGHTING** }

Functions

void **UGLIEnable** (**UGLIFeature** feature)
void **UGLIDisable** (**UGLIFeature** feature)

Detailed Description

Functionality to enable / disable various features.

Enumeration Type Documentation

enum UGLIFeature

Features that user can enable / disable, that will affect drawing.

Enumerator:

UGLI_DEPTH_TEST Whether or not each pixel should be tested for depth.
UGLI_BLEND Whether or not texture should be blended with color.
UGLI_LIGHTING Whether or not lighting should be used.

Function Documentation

void UGLIDisable (UGLIFeature *feature*)

Disables specified feature.

Parameters:

feature	Feature to disable.

void UGLIEnable (UGLIFeature *feature*)

Enables specified feature.

Parameters:

feature	Feature to enable.

---

gltransform.h File Reference

Stack-based transformation routines.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "glprimitive.h"
#include "matrixmath.h"
#include "angle.h"
#include "texture.h"
#include "header_begin.h"
#include "header_end.h"
```

Enumerations

enum **UGLIMatrixStackType** { **UGLI_MATRIX_PROJECTION**,
UGLI_MATRIX_MODELVIEW }

Used matrix types. enum **UGLIIndexType** { **UGLI_UBYTE**, **UGLI_USHORT**, **UGLI_UINT** }

Functions

void **UGLITransformGetMatrix** (float *matrix)
void **UGLITransformGetSpecifiedMatrix** (**UGLIMatrixStackType** stack, float *matrix)
void **UGLITransformFrustum** (float left, float right, float bottom, float top, float near, float far)
void **UGLITransformPerspective** (float fovy, float aspect, float near, float far)
void **UGLITransformOrtho** (float left, float right, float bottom, float top, float near, float far)
void **UGLITransformInitFrustum** (float left, float right, float bottom, float top, float near, float far)
void **UGLITransformInitPerspective** (float fovy, float aspect, float near, float far)
void **UGLITransformInitOrtho** (float left, float right, float bottom, float top, float near, float far)
void **UGLITransformMatrixMode** (**UGLIMatrixStackType** stack)
void **UGLITransformPushMatrix** (void)
void **UGLITransformPopMatrix** (void)
void **UGLITransformSetMatrixf** (const float *matrix)
void **UGLITransformMultMatrixf** (const float *matrix)
void **UGLITransformSetIdentity** (void)
void **UGLITransformTranslatef** (float x, float y, float z)
void **UGLITransformTranslate2f** (float x, float y)
void **UGLITransformScalef** (float x, float y, float z)
void **UGLITransformRotateAxis** (float angle, float x, float y, float z)
void **UGLITransformRotateX** (float angle)
void **UGLITransformRotateY** (float angle)
void **UGLITransformRotateZ** (float angle)
void **UGLISetVertexArrayPointers** (const float *pointer, int stride, const **UGLITexture** texture, unsigned int settings)
void **UGLISetPointerTexCoord2F** (int stride, const void *pointer)
void **UGLISetPointerColor4F** (int stride, const void *pointer)
void **UGLISetPointerNormal3F** (int stride, const void *pointer)
void **UGLISetPointerVertex4F** (int stride, const void *pointer)
void **UGLIDrawElements** (**UGLIPrimitive** primitive, int count, **UGLIIndexType** type, const void *indices)
void **UGLIDrawArrays** (**UGLIPrimitive** primitive, int first, int count)

Detailed Description

Stack-based transformation routines.

Enumeration Type Documentation

enum UGLIIndexType

Valid index types.

Enumerator:

UGLI_UBYTE Unsigned byte, all vertices must fit in a single byte.
UGLI_USHORT Unsigned short, all vertices must fit in 2 bytes.
UGLI_UINT Unsigned int, all vertices must fit in 4 bytes.

enum UGLIMatrixStackType

Used matrix types.

Enumerator:

UGLI_MATRIX_PROJECTION Projection matrix. Transforms 3D points to 2D points in viewport.
UGLI_MATRIX_MODELVIEW Modelview matrix. Transforms 3D points accordingly.

Function Documentation

void UGLIDrawArrays (UGLIPrimitive *primitive*, int *first*, int *count*)

Draws the specified amount of vertices as specified primitive from currently enabled arrays.

Parameters:

primitive	Which primitive to draw as.
first	First vertex to draw.
count	Amount of vertices to draw.

void UGLIDrawElements (UGLIPrimitive *primitive*, int *count*, UGLIIndexType *type*, const void * *indices*)

Draws the specified amount of indexed vertices as specified primitive from currently enabled arrays.

Parameters:

primitive	Which primitive to draw as.
count	Amount of vertices to draw.
type	Index format.
indices	Pointer to index array.

void UGLISetPointerColor4F (int *stride*, const void * *pointer*)

Sets color pointer.

Parameters:

stride	Amount of bytes between the start of each element.
pointer	Pointer to array that contains the values.

void UGLISetPointerNormal3F (int *stride*, const void * *pointer*)

Sets normal pointer.

Parameters:

stride	Amount of bytes between the start of each element.
pointer	Pointer to array that contains the values.

void UGLISetPointerTexCoord2F (int *stride*, const void * *pointer*)

Sets texture coordinate pointer.

Parameters:

stride	Amount of bytes between the start of each element.
pointer	Pointer to array that contains the values.

void UGLISetPointerVertex4F (int *stride*, const void * *pointer*)

Sets vertex pointer.

Parameters:

stride	Amount of bytes between the start of each element.
pointer	Pointer to array that contains the values.

void UGLISetVertexArrayPointers (const float * *pointer*, int *stride*, const UGLITexture *texture*, unsigned int *settings*)

Sets pointers accordingly for a draw operation.

Parameters:

pointer	Data pointer.
stride	Amount of bytes between the start of each vertex.
texture	Texture to use.
settings	Flags that define f.ex. which arrays to use. Order will always be the same.

void UGLITransformFrustum (float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Multiplies the current matrix in current stack with a matrix that corresponds to specified frustum.

Parameters:

left	Left clipping plane.
right	Right clipping plane.
bottom	Bottom clipping plane.
top	Top clipping plane.
near	Near clipping plane.
far	Far clipping plane.

void UGLITransformGetMatrix (float * *matrix*)

Gets the current matrix.

Parameters:

matrix	Space for 4*4 matrix.

void UGLITransformGetSpecifiedMatrix (UGLIMatrixStackType *stack*, float * *matrix*)

Gets the specified matrix.

Parameters:

stack	Which stack to get the matrix from.
matrix	Space for 4*4 matrix.

void UGLITransformInitFrustum (float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Initializes the current matrix in current stack to specified frustum.

Parameters:

left	Left clipping plane.
right	Right clipping plane.
bottom	Bottom clipping plane.
top	Top clipping plane.
near	Near clipping plane.
far	Far clipping plane.

void UGLITransformInitOrtho (float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Initializes the current matrix in current stack to specified orthogonal.

Parameters:

left	Left clipping plane.
right	Right clipping plane.
bottom	Bottom clipping plane.
top	Top clipping plane.
near	Near clipping plane.
far	Far clipping plane.

void UGLITransformInitPerspective (float *fovy*, float *aspect*, float *near*, float *far*)

Initializes the current matrix in current stack to specified perspective.

Parameters:

fovy	Field of vision.
aspect	Aspect ratio.
near	Near clipping plane.
far	Far clipping plane.

void UGLITransformMatrixMode (UGLIMatrixStackType *stack*)

Switches to the specified matrix stack.

Parameters:

stack	Matrix stack to switch to.

void UGLITransformMultMatrixf (const float * *matrix*)

Multiplies the matrix in current stack with the defined matrix.

Parameters:

matrix	Matrix to multiply stack matrix with.

void UGLITransformOrtho (float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Multiplies the current matrix in current stack with a matrix that corresponds to specified orthogonal.

Parameters:

left	Left clipping plane.
right	Right clipping plane.
bottom	Bottom clipping plane.
top	Top clipping plane.
near	Near clipping plane.
far	Far clipping plane.

void UGLITransformPerspective (float *fovy*, float *aspect*, float *near*, float *far*)

Multiplies the current matrix in current stack with a matrix that corresponds to specified perspective.

Parameters:

fovy	Field of vision.
aspect	Aspect ratio.
near	Near clipping plane.
far	Far clipping plane.

void UGLITransformPopMatrix (void)

Pops matrix from the current stack.

void UGLITransformPushMatrix (void)

Pushes matrix to the current stack.

void UGLITransformRotateAxis (float *angle*, float *x*, float *y*, float *z*)

Performs the specified axis angle rotation to the current matrix in the current stack.

Parameters:

angle	Rotation angle in degrees.
x	rotation X axis.
y	rotation Y axis.
z	rotation Z axis.

void UGLITransformRotateX (float *angle*)

Performs the specified rotation around X axis to the current matrix in the current stack.

Parameters:

angle	Rotation angle in degrees.

void UGLITransformRotateY (float *angle*)

Performs the specified rotation around Y axis to the current matrix in the current stack.

Parameters:

angle	Rotation angle in degrees.

void UGLITransformRotateZ (float *angle*)

Performs the specified rotation around Z axis to the current matrix in the current stack.

Parameters:

angle	Rotation angle in degrees.

void UGLITransformScalef (float *x*, float *y*, float *z*)

Scales the current matrix in the current stack.

Parameters:

x	X scale.
y	Y scale.
z	Z scale.

void UGLITransformSetIdentity (void)

Sets the current matrix in current stack to identity.

void UGLITransformSetMatrixf (const float * *matrix*)

Loads the specified matrix in the current stack.

Parameters:

matrix	Matrix to load.

void UGLITransformTranslate2f (float *x*, float *y*)

Translates the current matrix in the current stack.

Parameters:

x	X translation.
y	Y translation.

void UGLITransformTranslatef (float *x*, float *y*, float *z*)

Translates the current matrix in the current stack.

Parameters:

x	X translation.
y	Y translation.
z	Z translation.

---

image.h File Reference

Image handling.

```
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct UGLIImageData * **UGLIImage**

Functions

UGLIImage UGLINewImage (const char *filename)
void **UGLIDeleteImage** (**UGLIImage** image)
void **UGLIImageBlitToImage** (**UGLIImage** background, const **UGLIImage** foreground, int xoffset, int yoffset)

Detailed Description

Image handling.

Typedef Documentation

typedef struct UGLIImageData* UGLIImage

Image object.

Function Documentation

void UGLIDeleteImage (UGLIImage *image*)

Deletes specified image

Parameters:

image	Image object to delete

void UGLIImageBlitToImage (UGLIImage *background*, const UGLIImage *foreground*, int *xoffset*, int *yoffset*)

Blits specified foreground image on background.

Parameters:

background	Image to blit foreground image on.
foreground	Image to blit on background.
xoffset	X offset for foreground image.
yoffset	Y offset for foreground image.

UGLIImage UGLINewImage (const char * *filename*)

Loads specified image

Parameters:

filename	file to load image data from

Image object (or NULL on fail)

---

intrusivelinkedlist.h File Reference

Bi-directionally linked intrusive linked list.

```
#include <stdlib.h>
#include "memory.h"
#include "header_begin.h"
#include "header_end.h"
```

Data Structures

struct **UGLIIntrusiveLinkedListElement**
struct **UGLIIntrusiveLinkedList**

Typedefs

typedef struct
UGLIIntrusiveLinkedListElement UGLIIntrusiveLinkedListElement
typedef struct
UGLIIntrusiveLinkedList UGLIIntrusiveLinkedList

Functions

void **InitIntrusiveLinkedList** (**UGLIIntrusiveLinkedList** *list)
UGLIIntrusiveLinkedList * **NewIntrusiveLinkedList** (void)
void **DeleteIntrusiveLinkedList** (**UGLIIntrusiveLinkedList** *list)
void **IntrusiveLinkedListAppendElement** (**UGLIIntrusiveLinkedList** *list, **UGLIIntrusiveLinkedListElement** *element)
UGLIIntrusiveLinkedListElement * **IntrusiveLinkedListGetFirst** (**UGLIIntrusiveLinkedList** *list)
UGLIIntrusiveLinkedListElement * **IntrusiveLinkedListGetLast** (**UGLIIntrusiveLinkedList** *list)

UGLIIntrusiveLinkedListElement * **IntrusiveLinkedListRemoveFirst** (**UGLIIntrusiveLinkedList** *list)
UGLIIntrusiveLinkedListElement * **IntrusiveLinkedListRemoveLast** (**UGLIIntrusiveLinkedList** *list)
void **IntrusiveLinkedListRemoveElement** (**UGLIIntrusiveLinkedListElement** *element)
UGLIIntrusiveLinkedListElement * **IntrusiveLinkedListElementGetNextElement** (const **UGLIIntrusiveLinkedListElement** *current)
UGLIIntrusiveLinkedListElement * **IntrusiveLinkedListElementGetPreviousElement** (const **UGLIIntrusiveLinkedListElement** *current)
void **IntrusiveLinkedListElementInsertAfter** (**UGLIIntrusiveLinkedListElement** *current, **UGLIIntrusiveLinkedListElement** *next)
void **IntrusiveLinkedListElementInsertBefore** (**UGLIIntrusiveLinkedListElement** *current, **UGLIIntrusiveLinkedListElement** *previous)

Detailed Description

Bi-directionally linked intrusive linked list.

Typedef Documentation

typedef struct UGLIIntrusiveLinkedList UGLIIntrusiveLinkedList

Intrusive linked list struct.

typedef struct UGLIIntrusiveLinkedListElement UGLIIntrusiveLinkedListElement

Intrusive linked list element struct.

Function Documentation

void DeleteIntrusiveLinkedList (UGLIIntrusiveLinkedList * *list*)

Deletes the specified UGLIIntrusiveLinkedList.

Parameters:

Returns:

list	**UGLIIntrusiveLinkedList** to delete.

void InitIntrusiveLinkedList (UGLIIntrusiveLinkedList * *list*)

Initializes the specified UGLIIntrusiveLinkedList.

Parameters:

list	**UGLIIntrusiveLinkedList** to initialize.

void IntrusiveLinkedListAppendElement (UGLIIntrusiveLinkedList * *list*, UGLIIntrusiveLinkedListElement * *element*)

Appends specified UGLIIntrusiveLinkedListElement to the specified UGLIIntrusiveLinkedList.

Parameters:

list	**UGLIIntrusiveLinkedList** to append to.
element	**UGLIIntrusiveLinkedListElement** to append.

UGLIIntrusiveLinkedListElement* IntrusiveLinkedListElementGetNextElement (const UGLIIntrusiveLinkedListElement * *current*)

Gets next UGLIIntrusiveLinkedListElement.

Parameters:

current	Current **UGLIIntrusiveLinkedListElement**.

next **UGLIIntrusiveLinkedListElement** (or NULL if none).

UGLIIntrusiveLinkedListElement* IntrusiveLinkedListElementGetPreviousElement (const UGLIIntrusiveLinkedListElement * *current*)

Gets previous UGLIIntrusiveLinkedListElement.

Parameters:

Returns:

current	Current **UGLIIntrusiveLinkedListElement**.

previous **UGLIIntrusiveLinkedListElement** (or NULL if none).

void IntrusiveLinkedListElementInsertAfter (UGLIIntrusiveLinkedListElement * *current*, UGLIIntrusiveLinkedListElement * *next*)

Adds element to list after the specified element.

Parameters:

Returns:

current	Element, after which to insert the new element.
next	Element to insert.

void IntrusiveLinkedListElementInsertBefore (UGLIIntrusiveLinkedListElement * *current*, UGLIIntrusiveLinkedListElement * *previous*)

Adds element to list before the specified element.

Parameters:

current	Element, before which to insert the new element.
previous	Element to insert.

UGLIIntrusiveLinkedListElement* IntrusiveLinkedListGetFirst (UGLIIntrusiveLinkedList * *list*)

Gets the first UGLIIntrusiveLinkedListElement from the specified IntrusiveLinkedList.

Parameters:

list	**UGLIIntrusiveLinkedList** to get IntrusiveLinkedListElement from.

The first **UGLIIntrusiveLinkedListElement** (or NULL if none).

UGLIIntrusiveLinkedListElement* IntrusiveLinkedListGetLast (UGLIIntrusiveLinkedList * *list*)

Gets the last UGLIIntrusiveLinkedListElement from the specified IntrusiveLinkedList.

Parameters:

Returns:

list	**UGLIIntrusiveLinkedList** to get IntrusiveLinkedListElement from.

The last **UGLIIntrusiveLinkedListElement** (or NULL if none).

void IntrusiveLinkedListRemoveElement (UGLIIntrusiveLinkedListElement * *element*)

Removes the specified UGLIIntrusiveLinkedListElement from the list it belongs to.

Parameters:

Returns:

element	**UGLIIntrusiveLinkedListElement** to remove.

UGLIIntrusiveLinkedListElement* IntrusiveLinkedListRemoveFirst (UGLIIntrusiveLinkedList * *list*)

Removes first UGLIIntrusiveLinkedListElement from the specified IntrusiveLinkedList.

Parameters:

list	**UGLIIntrusiveLinkedList** to remove IntrusiveLinkedListElement from.

Removed **UGLIIntrusiveLinkedListElement** (or NULL if none).

UGLIIntrusiveLinkedListElement* IntrusiveLinkedListRemoveLast (UGLIIntrusiveLinkedList * *list*)

Removes last UGLIIntrusiveLinkedListElement from the specified IntrusiveLinkedList.

Parameters:

Returns:

list	**UGLIIntrusiveLinkedList** to remove IntrusiveLinkedListElement from.

Removed **UGLIIntrusiveLinkedListElement** (or NULL if none).

UGLIIntrusiveLinkedList* NewIntrusiveLinkedList (void)

Creates a new UGLIIntrusiveLinkedList.

Returns:

New **UGLIIntrusiveLinkedList**.

---

layout/layoutgrid.h File Reference

Grid layout object.

```
#include "layoutitem.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Functions

UGLILayoutItem UGLINewLayoutGrid (UGLILayoutItemFlag flags)

Detailed Description

Grid layout object.

Draws children in a grid. Empty rows and columns will be skipped.

Function Documentation

UGLILayoutItem UGLINewLayoutGrid (UGLILayoutItemFlag *flags*)

Constructor that creates a new grid layout.

Parameters:

Returns:

flags	flags.

new layout (if successful, otherwise NULL)

---

layout/layoutitem.h File Reference

Core UGLILayoutItem object.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
```

```
#include "../uielements/glframe.h"
#include "../events.h"
#include "../reallocarray.h"
#include "../time.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Macros

#define **UGLI_LI_SIZE_MAX_NO_LIMIT** (-1)

No maximum limit, item can grow as big as there's space left.

#define **UGLI_LI_ALIGN_MIN** 0.0f
#define **UGLI_LI_ALIGN_CEN** 0.5f
#define **UGLI_LI_ALIGN_MAX** 1.0f
#define **UGLI_LI_ALIGN_PRV** (-1.0f)

Typedefs

typedef struct **UGLILayoutItemData * UGLILayoutItem**

UGLILayoutItem struct.

Enumerations

enum **UGLILayoutItemDrawFlag** { **UGLI_LI_CMD_F_DRAW_NONE** = 0,
UGLI_LI_CMD_F_DRAW_3D_LEFT = 0x10,
UGLI_LI_CMD_F_DRAW_3D_RIGHT = 0x20 }

Gives UGLILayoutItem extra information about what to draw. enum **UGLILayoutItemEventHandleFlag** { **UGLI_LI_EVENTHANDLE_F_NONE** = 0x00, **UGLI_LI_EVENTHANDLE_F_HANDLED** = 0x01 }

enum **UGLILayoutItemAddChildError** { **UGLI_LI_ADD_CHILD_NO_ERROR**,
UGLI_LI_ADD_CHILD_NULL_POINTER,
UGLI_LI_ADD_CHILD_PARENT_REFUSED,
UGLI_LI_ADD_CHILD_HAS_PARENT,
UGLI_LI_ADD_CHILD_SLOT_OCCUPIED,
UGLI_LI_ADD_CHILD_UNDEFINED }

Error values for functions for adding children to UGLILayoutItems. enum **UGLILayoutItemFlag** { **UGLI_LI_F_NONE** = 0x0000, **UGLI_LI_F_ACCEPTS_INPUT** = 0x0008, **UGLI_LI_F_SET_VIEWPORT** = 0x0010, **UGLI_LI_F_HIDEABLE_CHILDREN** = 0x0040, **UGLI_LI_F_CACHE_CONTENTS** = 0x0400, **UGLI_LI_F_NEEDS_ORTHO** = 0x0800, **UGLI_LI_F_CREATE_PROJECTION** = 0x1000 }

Flags you can pass to UGLILayoutItem constructor. enum **UGLILayoutItemDrawStatusFlag** { **UGLI_LI_F_DRAW_NONE** = 0x0000, **UGLI_LI_F_DRAW_DEPTH_BUFFER** = 0x0001, **UGLI_LI_F_DRAW_COLOR_BUFFER** = 0x0002, **UGLI_LI_F_DRAW_NEEDS_UPDATE** = 0x0004 }

Returned by ->Draw(). Defines which buffers the item has drawn to. enum **UGLILayoutItemClearMode** { **UGLI_LI_CLEAR_NOTHING** = 0, **UGLI_LI_CLEAR_DEPTH** = 1, **UGLI_LI_CLEAR_COLOR** = 2, **UGLI_LI_CLEAR_ALL** = 3 }

Defines what UGLILayoutItem expects to be cleared, before being drawn. enum **UGLILayoutItemWeight** { **UGLI_LI_WEIGHT_HORIZONTAL** = 0x01, **UGLI_LI_WEIGHT_VERTICAL** = 0x02, **UGLI_LI_WEIGHT_BOTH** = 0x03 }

Specifies which weight(s) of the UGLILayoutItem will be modified. You can set either one, or both. Weight defines, how much of the available extra space the UGLILayoutItem will get, compared to the others. Default is 50. enum **UGLILayoutSizeHint** { **UGLI_LI_SIZE_MINIMUM** = 1, **UGLI_LI_SIZE_MAXIMUM** = 2, **UGLI_LI_SIZE_PREFERRED** = 4, **UGLI_LI_SIZE_STATIC** = 7 }

Enum to let user specify, which UGLILayoutItem size hint he wants to modify. enum **UGLILayoutMargin** { **UGLI_LI_MARGIN_NONE** = 0x00, **UGLI_LI_MARGIN_LEFT** = 0x01, **UGLI_LI_MARGIN_RIGHT** = 0x02, **UGLI_LI_MARGIN_HORIZONTAL** = 0x03, **UGLI_LI_MARGIN_TOP** = 0x04, **UGLI_LI_MARGIN_BOTTOM** = 0x08, **UGLI_LI_MARGIN_VERTICAL** = 0x0C, **UGLI_LI_MARGIN_ALL** = 0x0F }

Enum to let user specify, which UGLILayoutItem margin he wants to modify.

Functions

void **UGLILayoutItemSetSizeHint** (**UGLILayoutItem** layoutitem, **UGLILayoutSizeHint** sizehint, int x, int y)

void **UGLILayoutItemSetMargin** (**UGLILayoutItem** layoutitem, **UGLILayoutMargin** which, unsigned int value)

void **UGLILayoutItemSetMargins** (**UGLILayoutItem** layoutitem, unsigned int left, unsigned int right, unsigned int top, unsigned int bottom)

void **UGLILayoutItemSetFrame** (**UGLILayoutItem** layoutitem, **UGLIUIFrame** frame)

void **UGLILayoutItemSetClearMode** (**UGLILayoutItem** layoutitem, **UGLILayoutItemClearMode** clearmode)

void **UGLILayoutItemSetVisibility** (**UGLILayoutItem** layoutitem, int visibility)

UGLILayoutItemDrawStatusFlag UGLILayoutItemDrawRoot (**UGLILayoutItem** layoutitem, **UGLITicks** delta, **UGLILayoutItemDrawFlag** drawflags)

UGLILayoutItem UGLINewLayoutItem (void(*Delete)(**UGLILayoutItem** layoutitem), **UGLILayoutItemDrawStatusFlag**(*Draw)(const **UGLILayoutItem** layoutitem, **UGLITicks** delta, **UGLILayoutItemDrawFlag** drawflags), void(*ApplyViewport)(**UGLILayoutItem** layoutitem), void(*UpdateSizeHint)(**UGLILayoutItem** layoutitem), void(*ChildDeleted)(**UGLILayoutItem** layoutitem, const **UGLILayoutItem** childli), **UGLILayoutItem**(*WhichLayoutItem)(**UGLILayoutItem** layoutitem, **UGLIEventType** type, int x, int y), **UGLILayoutItemEventHandleFlag**(*HandleEvent)(**UGLILayoutItem** layoutitem, const **UGLIEventPointer** event), **UGLILayoutItemClearMode** clearmode, void *data, **UGLILayoutItemFlag** flags)

void **UGLIDeleteLayoutItem** (**UGLILayoutItem** layoutitem)

void **UGLILayoutItemSetAlignment** (**UGLILayoutItem** layoutitem, float xalignment, float yalignment)

void **UGLILayoutItemSetWeight** (**UGLILayoutItem** layoutitem, **UGLILayoutItemWeight** which, float weight)

void **UGLILayoutItemUpdateSizeHint** (**UGLILayoutItem** layoutitem)

UGLILayoutItemAddChildError UGLILayoutItemAddItem (**UGLILayoutItem** parent, **UGLILayoutItem** child)
UGLILayoutItemAddChildError UGLILayoutItemAddItemAt (**UGLILayoutItem** parent, **UGLILayoutItem** child,...)
int **UGLILayoutItemCoversPixel** (const **UGLILayoutItem** layoutitem, int x, int y)
void * **UGLILayoutItemGetData** (**UGLILayoutItem** layoutitem)
UGLILayoutItemEventHandleFlag UGLILayoutItemHandleEvent (**UGLILayoutItem** layoutitem, const **UGLIEventPointer** event)
void **UGLILayoutItemSetAddChild** (**UGLILayoutItem** layoutitem, **UGLILayoutItemAddChildError**(*AddItem)(**UGLILayoutItem** layoutitem, **UGLILayoutItem** item), **UGLILayoutItemAddChildError**(*AddItemAt) (**UGLILayoutItem** layoutitem, **UGLILayoutItem** item, va_list args))
void **UGLILayoutItemBeginUpdate** (**UGLILayoutItem** layoutitem)
void **UGLILayoutItemEndUpdate** (**UGLILayoutItem** layoutitem)
void **UGLILayoutItemRequestRedraw** (**UGLILayoutItem** layoutitem)
void **UGLILayoutItemSetMouseMotionAccept** (**UGLILayoutItem** layoutitem, int accept)

Detailed Description

Core UGLILayoutItem object.

Objects using UGLILayoutItem should create a new UGLILayoutItem object with NewLayoutItem constructor, passing a pointer to their own private data as data pointer.

Macro Definition Documentation

#define UGLI_LI_ALIGN_CEN 0.5f

Align to center.

#define UGLI_LI_ALIGN_MAX 1.0f

Align to "maximum", right/bottom depending on the direction.

#define UGLI_LI_ALIGN_MIN 0.0f

Align to "minimum", left/top depending on the direction.

#define UGLI_LI_ALIGN_PRV (-1.0f)

No change. Use the previous alignment value for this direction.

#define UGLI_LI_SIZE_MAX_NO_LIMIT (-1)

No maximum limit, item can grow as big as there's space left.

No restriction for item size. Item can use all available extra space in dimensions, where it's set. Extra space will be shared between all items that can grow beyond minimum according to their weight, and then between all UGLI_LI_SIZE_MAX_NO_LIMIT items, according to their weight.

Typedef Documentation

typedef struct UGLILayoutItemData* UGLILayoutItem

UGLILayoutItem struct.

It's not needed to know the contents for most functionality specified in this file. If you are creating your own UGLILayoutItem, then you probably need to know some of the internals. You can find them in **layoutitem_struct.h**.

Enumeration Type Documentation

enum UGLILayoutItemAddChildError

Error values for functions for adding children to UGLILayoutItems.

Many UGLILayoutItems don't support children, it's a good idea to check the documentation if the one used does, rather than depend on the error value. Each UGLILayoutItem "child-slot" can have only one child. If you try adding one on top of an existing one (with *At() function), old one will be kept, and an error value will be returned.

Enumerator:

UGLI_LI_ADD_CHILD_NO_ERROR Child added successfully.

UGLI_LI_ADD_CHILD_NULL_POINTER Error: One or more pointers were NULL.

UGLI_LI_ADD_CHILD_PARENT_REFUSED Error: Parent refused to add child.

UGLI_LI_ADD_CHILD_HAS_PARENT Error: Child already has parent.

UGLI_LI_ADD_CHILD_SLOT_OCCUPIED Error: Specified slot already has a child.

UGLI_LI_ADD_CHILD_UNDEFINED Error: Undefined error.

enum UGLILayoutItemClearMode

Defines what UGLILayoutItem expects to be cleared, before being drawn.

Defines, which buffers should be clear, before the specific UGLILayoutItem is drawn. Only the area under UGLILayoutItem is considered, so if one UGLILayoutItem draws to depth buffer and another one expects clear depth buffer before being drawn, it won't be needed to be cleared again, as long as it was done before the first object, and the two items don't overlap.

Enumerator:

UGLI_LI_CLEAR_NOTHING No need to clear anything.
UGLI_LI_CLEAR_DEPTH UGLILayoutItem expects clear depth buffer.
UGLI_LI_CLEAR_COLOR UGLILayoutItem expects clear color buffer.
UGLI_LI_CLEAR_ALL UGLILayoutItem expects clear depth and color buffers.

enum UGLILayoutItemDrawFlag

Gives UGLILayoutItem extra information about what to draw.

Flags, that the library might pass your UGLILayoutItem. It's up to code to decide, whether or not to care about them. Currently, this mostly acts as a placeholder for some kind of 3D mode.

Enumerator:

UGLI_LI_CMD_F_DRAW_NONE Just draw normally..
UGLI_LI_CMD_F_DRAW_3D_LEFT Draw "left eye" image.
UGLI_LI_CMD_F_DRAW_3D_RIGHT Draw "right eye" image.

enum UGLILayoutItemDrawStatusFlag

Returned by ->Draw(). Defines which buffers the item has drawn to.

This information is used when deciding, whether there's need for any clearing before drawing any UGLILayoutItems This is used in combination with flags that define, whether an item needs a clear deph buffer before drawing.

Enumerator:

UGLI_LI_F_DRAW_NONE Item didn't draw anything anywhere.
UGLI_LI_F_DRAW_DEPTH_BUFFER Item has drawn to the depth buffer.
UGLI_LI_F_DRAW_COLOR_BUFFER Item has drawn to the color buffer.
UGLI_LI_F_DRAW_NEEDS_UPDATE Item requests window redraw.

enum UGLILayoutItemEventHandleFlag

Enumerator:

UGLI_LI_EVENTHANDLE_F_NONE No flags.
UGLI_LI_EVENTHANDLE_F_HANDLED Event handled by UGLILayoutItem. No further processing needed.

enum UGLILayoutItemFlag

Flags you can pass to UGLILayoutItem constructor.

Most critical are LIFLAG_NEEDS_ORTHO and LIFLAG_SET_VIEWPORT (if needed). Hidden by priority / size shouldn't be touched.

Enumerator:

UGLI_LI_F_NONE No flags.

UGLI_LI_F_SET_VIEWPORT If set, UGLILayoutItem needs viewport to be set to UGLILayoutItem size before ->Draw() call. It's preferable to avoid this flag whenever possible, but if omitted, your item must be able to restrict drawing to the area assigned for it.
UGLI_LI_F_CACHE_CONTENTS Requests content caching (FBO) - currently unimplemented.
UGLI_LI_F_NEEDS_ORTHO If set, UGLILayoutItem needs projection matrix set to orthogonal projection.
UGLI_LI_F_CREATE_PROJECTION Create projection matrix for object, that will be automatically set before drawing. Will be initialized to identity.

enum UGLILayoutItemWeight

Specifies which weight(s) of the UGLILayoutItem will be modified. You can set either one, or both. Weight defines, how much of the available extra space the UGLILayoutItem will get, compared to the others. Default is 50.

When assigning the extra space, total weight of all components should be counted together, and each UGLILayoutItems' weight compared againist that.

Enumerator:

UGLI_LI_WEIGHT_HORIZONTAL Set horizontal weight.
UGLI_LI_WEIGHT_VERTICAL Set vertical weight.
UGLI_LI_WEIGHT_BOTH Set both weights.

enum UGLILayoutMargin

Enum to let user specify, which UGLILayoutItem margin he wants to modify.

Values can also be OR'ed (|) together.

Enumerator:

UGLI_LI_MARGIN_NONE None, don't modify any margin.
UGLI_LI_MARGIN_LEFT Set the left margin.
UGLI_LI_MARGIN_RIGHT Set the right margin.
UGLI_LI_MARGIN_HORIZONTAL Set both right and left margins.
UGLI_LI_MARGIN_TOP Set the top margin.
UGLI_LI_MARGIN_BOTTOM Set the bottom margin.
UGLI_LI_MARGIN_VERTICAL Set both top and bottom margins.
UGLI_LI_MARGIN_ALL Set all margins.

enum UGLILayoutSizeHint

Enum to let user specify, which UGLILayoutItem size hint he wants to modify.

Enumerator:

UGLI_LI_SIZE_MINIMUM Minimum size, contents don't want to be smaller than this in any case.

UGLI_LI_SIZE_MAXIMUM Contents don't want to be bigger than this.
UGLI_LI_SIZE_PREFERRED "Comfortable size", layout tries to give at least this amount to each item.
UGLI_LI_SIZE_STATIC Contents want to be always exactly this size. Sets both min and max.

Function Documentation

void UGLIDeleteLayoutItem (UGLILayoutItem *layoutitem*)

Destructor that deletes the specified UGLILayoutItem along with subclass (if destructor for it was specified).

Parameters:

Returns:

layoutitem	UGLILayoutItem to delete UGLILayoutItem destructor should NOT free UGLILayoutItem itself, only li->data!

UGLILayoutItemAddChildError UGLILayoutItemAddItem (UGLILayoutItem *parent*, UGLILayoutItem *child*)

Add single child UGLILayoutItem at point chosen by parent.

Parameters:

parent	Parent UGLILayoutItem to add child to.
child	Child UGLILayoutItem to add.

error (or LI_ADD_CHILD_NO_ERROR if successful)

UGLILayoutItemAddChildError UGLILayoutItemAddItemAt (UGLILayoutItem *parent*, UGLILayoutItem *child*, ...)

Add single child UGLILayoutItem at point defined by arguments.

Parameters:

Returns:

parent	Parent UGLILayoutItem to add child to.
child	Child UGLILayoutItem to add.
...	position to add child to.

error (or LI_ADD_CHILD_NO_ERROR if successful)

void UGLILayoutItemBeginUpdate (UGLILayoutItem *layoutitem*)

Begins series of updates. Changes won't trigger redraw until changes are ended.

Parameters:

Returns:

layoutitem	UGLILayoutItem the changes concern.

int UGLILayoutItemCoversPixel (const UGLILayoutItem *layoutitem*, int *x*, int *y*)

Tells whether or not given UGLILayoutItem covers specified pixel.

Parameters:

layoutitem	UGLILayoutItem to check.
x	X position of the pixel.
y	Y position of the pixel.

1 if UGLILayoutItem covers the pixel, 0 otherwise (or if UGLILayoutItem==NULL)

UGLILayoutItemDrawStatusFlag UGLILayoutItemDrawRoot (UGLILayoutItem *layoutitem*, UGLITicks *delta*, UGLILayoutItemDrawFlag *drawflags*)

Draws the specified UGLILayoutItem as root.

Parameters:

Returns:

layoutitem	Which UGLILayoutItem to draw.
delta	Time since program start.
drawflags	Drawflags.

void UGLILayoutItemEndUpdate (UGLILayoutItem *layoutitem*)

Ends series of updates. If there were changes that require redraw, it's done now.

Parameters:

layoutitem	UGLILayoutItem the changes concern.

void* UGLILayoutItemGetData (UGLILayoutItem *layoutitem*)

Returns UGLILayoutItem data pointer.

Parameters:

layoutitem	UGLILayoutItem to get data from.

UGLILayoutItem data pointer, if defined & li != NULL.

UGLILayoutItemEventHandleFlag UGLILayoutItemHandleEvent (UGLILayoutItem *layoutitem*, const UGLIEventPointer *event*)

Gives event to UGLILayoutItem. If it isn't handled (not interested, or missing handler), returns 0.

Parameters:

Returns:

layoutitem	UGLILayoutItem to send event to.
event	Event to handle.

0 if not handled, otherwise something else.

void UGLILayoutItemRequestRedraw (UGLILayoutItem *layoutitem*)

Request layoutitem redraw.

Parameters:

Returns:

layoutitem	UGLILayoutItem requested for redraw.

void UGLILayoutItemSetAddChild (UGLILayoutItem *layoutitem*, UGLILayoutItemAddChildError(*)(UGLILayoutItem layoutitem, UGLILayoutItem item) *AddItem*, UGLILayoutItemAddChildError(*)(UGLILayoutItem layoutitem, UGLILayoutItem item, va_list args) *AddItemAt*)

Sets UGLILayoutItem item addition functions.

Parameters:

layoutitem	UGLILayoutItem, for which to add the AddItem() functions.
AddItem	Function to add single child UGLILayoutItem at point chosen by parent (or NULL for none).
AddItemAt	Function to add single child UGLILayoutItem at point defined by arguments (or NULL for none).

void UGLILayoutItemSetAlignment (UGLILayoutItem *layoutitem*, float *xalignment*, float *yalignment*)

Sets UGLILayoutItem alignments to specified values.

Parameters:

layoutitem	UGLILayoutItem to set alignment for.
xalignment	X alignment, ALIGN_MIN, ALIGN_CEN, ALIGN_MAX or any float 0.0f-1.0f. ALIGN_PRV keeps the previously set value.
yalignment	Y alignment, ALIGN_MIN, ALIGN_CEN, ALIGN_MAX or any float 0.0f-1.0f. ALIGN_PRV keeps the previously set value.

void UGLILayoutItemSetClearMode (UGLILayoutItem *layoutitem*, UGLILayoutItemClearMode *clearmode*)

Sets how LI expects its background to be before being drawn (in other words, what to glClear()).

Parameters:

layoutitem	Which UGLILayoutItem to set the clear mode for.
clearmode	New clearmode.

void UGLILayoutItemSetFrame (UGLILayoutItem *layoutitem*, UGLIUIFrame *frame*)

Sets all UGLILayoutItem frame.

Parameters:

layoutitem	Which UGLILayoutItem to set the frame for.
frame	New frame (or NULL for none).

void UGLILayoutItemSetMargin (UGLILayoutItem *layoutitem*, UGLILayoutMargin *which*, unsigned int *value*)

Sets specified UGLILayoutItem margin(s)

Parameters:

layoutitem	Which UGLILayoutItem to set the margin for.
which	Which margin(s) to set.
value	New margin value.

void UGLILayoutItemSetMargins (UGLILayoutItem *layoutitem*, unsigned int *left*, unsigned int *right*, unsigned int *top*, unsigned int *bottom*)

Sets all UGLILayoutItem margins separately.

Parameters:

layoutitem	Which UGLILayoutItem to set the margins for.
left	New left margin value.
right	New right margin value.
top	New top margin value.
bottom	New bottom margin value.

void UGLILayoutItemSetMouseMotionAccept (UGLILayoutItem *layoutitem*, int *accept*)

Sets whether or not to accept mouse motion events.

Parameters:

layoutitem	UGLILayoutItem to set.
accept	Whether or not to accept mouse motion events.

void UGLILayoutItemSetSizeHint (UGLILayoutItem *layoutitem*, UGLILayoutSizeHint *sizehint*, int *x*, int *y*)

Sets specified UGLILayoutItem SizeHint.

Parameters:

layoutitem	Which UGLILayoutItem to get size hint from.
sizehint	Which SizeHint to set
x	Width of size hint.
y	Height of size hint.

void UGLILayoutItemSetVisibility (UGLILayoutItem *layoutitem*, int *visibility*)

Sets whether or not the UGLILayoutItem is visible.

Parameters:

layoutitem	Which UGLILayoutItem to set the visibility for.
visibility	New visibility state.

void UGLILayoutItemSetWeight (UGLILayoutItem *layoutitem*, UGLILayoutItemWeight *which*, float *weight*)

Sets UGLILayoutItem weight.

Parameters:

layoutitem	UGLILayoutItem to set weight for.

which	Defines which weight(s) to set.
weight	New weight value.

void UGLILayoutItemUpdateSizeHint (UGLILayoutItem *layoutitem*)

Requests UGLILayoutItem to update its size hints.

Parameters:

layoutitem	UGLILayoutItem to update.

UGLILayoutItem UGLINewLayoutItem (void(*)(UGLILayoutItem layoutitem) *Delete*, UGLILayoutItemDrawStatusFlag(*)(const UGLILayoutItem layoutitem, UGLITicks delta, UGLILayoutItemDrawFlag drawflags) *Draw*, void(*)(UGLILayoutItem layoutitem) *ApplyViewport*, void(*)(UGLILayoutItem layoutitem) *UpdateSizeHint*, void(*)(UGLILayoutItem layoutitem, const UGLILayoutItem childli) *ChildDeleted*, UGLILayoutItem(*)(UGLILayoutItem layoutitem, UGLIEventType type, int x, int y) *WhichLayoutItem*, UGLILayoutItemEventHandleFlag(*)(UGLILayoutItem layoutitem, const UGLIEventPointer event) *HandleEvent*, UGLILayoutItemClearMode *clearmode*, void * *data*, UGLILayoutItemFlag *flags*)

Constructor that creates a new UGLILayoutItem object suitable for subclasses.

Parameters:

Delete	Destructor for subclass.
Draw	Member function used to draw subclass. When called, viewport & projection matrix are set as requested, and modelview matrix stack selected with identity matrix on top.
ApplyViewport	Memeber function that's executed after UGLILayoutItem vieport changes.
UpdateSizeHint	Member function that updates size requirements (f.ex. after adding child).
ChildDeleted	Member function that's called when a child is deleted.
WhichLayoutItem	Member function that returns, which child UGLILayoutItem is under given coordinates.
HandleEvent	Member function that handles events.
clearmode	Defines, if UGLILayoutItem requires clear depth / color buffer
data	pointer to the subclass (optional)
flags	Flags

new UGLILayoutItem (if successful, otherwise NULL)

---

layout/layoutitem_struct.h File Reference

LayoutItem struct.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "../uielements/glframe.h"
#include "../events.h"
#include "../reallocarray.h"
#include "../time.h"
```

```
#include "../flag_p.h"
#include "layoutitem.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Data Structures

struct **UGLILayoutItemData**

LayoutItem struct for code that implements new UGLILayoutItem. **Typedefs**

typedef struct
UGLILayoutItemPrivateData * **UGLILayoutItemPrivate**

LayoutItem private struct with items, user doesn't need to know.

Enumerations

enum **UGLILayoutItemHidden** { **UGLI_LI_F_VISIBLE** = 0x0000,
UGLI_LI_F_HIDDEN_BY_USER = UGLI_FLAG(0),
UGLI_LI_F_HIDDEN_BY_PRIORITY = UGLI_FLAG(1),
UGLI_LI_F_HIDDEN_BY_SIZE = UGLI_FLAG(2) }

Functions

UGLILayoutItemDrawStatusFlag
UGLIDrawLayoutItemsReallocTableNoOverlap (const **UGLIReallocArray** *child, **UGLITicks** delta, **UGLILayoutItemDrawFlag** drawflags)

Detailed Description

LayoutItem struct.

Public contents of LayoutItem struct. Only needed for files that define derived objects.

Typedef Documentation

typedef struct UGLILayoutItemPrivateData* UGLILayoutItemPrivate

LayoutItem private struct with items, user doesn't need to know.

All items the user doesn't need to know are stored inside this struct.

Enumeration Type Documentation

enum UGLILayoutItemHidden

Enumerator:

UGLI_LI_F_VISIBLE UGLILayoutItem is visible.
UGLI_LI_F_HIDDEN_BY_USER UGLILayoutItem has been hidden by user.

UGLI_LI_F_HIDDEN_BY_PRIORITY UGLILayoutItem has been hidden by priority (mostly unimplemented currently).
UGLI_LI_F_HIDDEN_BY_SIZE UGLILayoutItem has been hidden due to lack of space.

Function Documentation

UGLILayoutItemDrawStatusFlag UGLIDrawLayoutItemsReallocTableNoOverlap (const UGLIReallocArray * *child*, UGLITicks *delta*, UGLILayoutItemDrawFlag *drawflags*)

Draws ReallocTable full of UGLILayoutItems which are guaranteed to not overlap, in the most efficient way.

Parameters:

Returns:

child	List of UGLILayoutItems to draw.
delta	Time since program start.
drawflags	Drawflags.

---

layout/layoutlinear.h File Reference

Linear layout object.

```
#include "layoutitem.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Enumerations

enum **UGLILayoutLinearDirection** { **UGLI_LI_DIRECTION_HORIZONTAL**, **UGLI_LI_DIRECTION_VERTICAL** }
Defines linear layout direction. **Functions**

UGLILayoutItem UGLINewLayoutLinear (**UGLILayoutLinearDirection** direction, **UGLILayoutItemFlag** flags)
UGLILayoutItem UGLINewLayoutVertical (**UGLILayoutItemFlag** flags)
UGLILayoutItem UGLINewLayoutHorizontal (**UGLILayoutItemFlag** flags)

Detailed Description

Linear layout object.

UGLILayoutItem that lays children in horizontal or vertical direction in a single row.

Enumeration Type Documentation

enum UGLILayoutLinearDirection

Defines linear layout direction.

UGLI_LI_DIRECTION_HORIZONTAL means, that child items will be stored in a horizontal row, while UGLI_LI_DIRECTION_VERTICAL means the row will be vertical instead.

First item will be top / left, depending on the direction. In the second dimension, LayoutItems will be drawn according to their alignment.

Enumerator:

UGLI_LI_DIRECTION_HORIZONTAL Left-to-right.
UGLI_LI_DIRECTION_VERTICAL Up-to-down.

Function Documentation

UGLILayoutItem UGLINewLayoutHorizontal (UGLILayoutItemFlag *flags*)

Constructor that creates a new horizontal layout.

Parameters:

flags	flags.

new LayoutItem (if successful, otherwise NULL)

UGLILayoutItem UGLINewLayoutLinear (UGLILayoutLinearDirection *direction*, UGLILayoutItemFlag *flags*)

Constructor that creates a new linear layout.

Parameters:

Returns:

direction	Direction the layout will flow.
flags	flags.

new LayoutItem (if successful, otherwise NULL)

UGLILayoutItem UGLINewLayoutVertical (UGLILayoutItemFlag *flags*)

Constructor that creates a new vertical layout.

Parameters:

Returns:

flags	flags.

new LayoutItem (if successful, otherwise NULL)

layout/layoutstack.h File Reference

Stack layout object.

```
#include "layoutitem.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Functions

UGLILayoutItem UGLINewLayoutStack (**UGLILayoutItemFlag** flags)

Detailed Description

Stack layout object.

Layout, in which children are drawn on top of each other. Good for HUD displays.

Function Documentation

UGLILayoutItem UGLINewLayoutStack (UGLILayoutItemFlag *flags*)

Constructor that creates a new stack layout.

Parameters:

Returns:

flags	flags.

new LayoutItem (if successful, otherwise NULL)

---

layout/layoutswitch.h File Reference

Switch LayoutItem object.

```
#include "../header_begin.h"
#include "layoutitem.h"
#include "../header_end.h"
```

Functions

UGLILayoutItem UGLINewLayoutSwitch (**UGLILayoutItemFlag** flags)

void **UGLILayoutSwitchSetActiveItem** (**UGLILayoutItem** layoutitem,
UGLILayoutItem child)

Detailed Description

Switch LayoutItem object.

LayoutItem that can contain several children, of which only one can be visible at once.

Function Documentation

void UGLILayoutSwitchSetActiveItem (UGLILayoutItem *layoutitem*, UGLILayoutItem *child*)

Sets active item for the specified LayoutSwitch.

Parameters:

Returns:

layoutitem	Parent item.
child	New active item.

UGLILayoutItem UGLINewLayoutSwitch (UGLILayoutItemFlag *flags*)

Constructor that creates a new switch layout.

Parameters:

flags	flags.

new LayoutItem (if successful, otherwise NULL)

---

layout/layoutview.h File Reference

LayoutView object.

```
#include "../physics.h"
#include "../window.h"
#include "layoutitem.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Typedefs

typedef struct UGLILayoutViewData * **UGLILayoutView**
UGLILayoutView object.

Functions

UGLILayoutView UGLINewLayoutView (**UGLILayoutItem** contents, **UGLIPhysics** physics)
void **UGLIDeleteLayoutView** (**UGLILayoutView** layoutview)
void **UGLILayoutViewSetContents** (**UGLILayoutView** layoutview, **UGLILayoutItem** contents)

void **UGLILayoutViewSetKeyboardEventHandler** (**UGLILayoutView** layoutview, **UGLILayoutItem** layoutitem)
void **UGLILayoutViewReshape** (void *data, int width, int height)
void **UGLILayoutViewDraw** (void *data)
int **UGLILayoutViewRunEventLoop** (**UGLILayoutView** layoutview)
void **UGLILayoutViewSendUserEvent** (**UGLILayoutView** layoutview, **UserEventType** type, void *data1, void *data2)
void **UGLILayoutViewQuitEventLoop** (void)
void **UGLILayoutViewSetUserEventHandler** (**UGLILayoutView** layoutview, **UGLIEventHandleFlag**(*UserEventHandler)(void *object, const **UGLIUserEvent** *e), void *object)

Detailed Description

LayoutView object.

"Root" item. Between Window and LayoutItem. Passes events etc.

Typedef Documentation

typedef struct UGLILayoutViewData* UGLILayoutView

UGLILayoutView object.

LayoutView is the interface between the window and root LayoutItem.

Function Documentation

void UGLIDeleteLayoutView (UGLILayoutView *layoutview*)

Destructor that deletes the specified LayoutView.

Parameters:

Returns:

layoutview	LayoutView to delete.

void UGLILayoutViewDraw (void * *data*)

Draw callback for window.

Parameters:

data	Object pointer.

void UGLILayoutViewQuitEventLoop (void)

Signals UGLILayoutView to quit event loop.

void UGLILayoutViewReshape (void * *data*, int *width*, int *height*)

Reshape callback for window.

Parameters:

data	Object pointer.
width	Window width.
height	Window height.

int UGLILayoutViewRunEventLoop (UGLILayoutView *layoutview*)

Runs UGLILayoutView event loop.

Parameters:

layoutview	UGLILayoutView for which to run the event loop.

Status.

void UGLILayoutViewSendUserEvent (UGLILayoutView *layoutview*, UserEventType *type*, void * *data1*, void * *data2*)

Sends a user event to the specified UGLILayoutView.

Parameters:

Returns:

layoutview	UGLILayoutView to send user event to.
type	User event type.
data1	Data pointer 1.
data2	Data pointer 2.

void UGLILayoutViewSetContents (UGLILayoutView *layoutview*, UGLILayoutItem *contents*)

Sets LayoutView contents.

Parameters:

layoutview	UGLILayoutView to set contents for.
contents	LayoutItem that will be used as contents (or NULL for none).

void UGLILayoutViewSetKeyboardEventHandler (UGLILayoutView *layoutview*, UGLILayoutItem *layoutitem*)

Sets LayoutView keyboard handler.

Parameters:

layoutview	UGLILayoutView to set keyboard handler for.
layoutitem	to send keyboard events to.

void UGLILayoutViewSetUserEventHandler (UGLILayoutView *layoutview*,
UGLIEventHandleFlag(*)(void *object, const UGLIUserEvent *e)
***UserEventHandler*, void * *object*)**

Sets UGLILayoutView user event handler.

Parameters:

layoutview	UGLILayoutView to set user event handler for.
UserEventHandler	New user event handler (or NULL for none).
object	Object pointer.

UGLILayoutView UGLINewLayoutView (UGLILayoutItem *contents*, UGLIPhysics *physics*)

Constructor that creates a new LayoutView.

Parameters:

contents	LayoutItem that will be used as contents (or NULL for none).
physics	A physics handler (or NULL for none).

new LayoutItem (if successful, otherwise NULL)

---

layout/spacer.h File Reference

Spacer LayoutItem object.

```
#include "layoutitem.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Enumerations

enum **SpacerDirectionFlag** {
UGLI_LI_F_SPACER_DIRECTION_HORIZONTAL = 0x01,
UGLI_LI_F_SPACER_DIRETION_VERTICAL = 0x02,
UGLI_LI_F_SPACER_DIRECTION_BOTH = 0x03 }
Spacer direction. **Functions**

UGLILayoutItem NewSpacer (**UGLIWindow** window, **SpacerDirectionFlag** flags)

Detailed Description

Spacer LayoutItem object.

Takes specified amount of space, won't draw anything.

Enumeration Type Documentation

enum SpacerDirectionFlag

Spacer direction.

Defines which direction(s) the spacer should expand in.

Enumerator:

UGLI_LI_F_SPACER_DIRECTION_HORIZONTAL Spacer should expand horizontally.

UGLI_LI_F_SPACER_DIRETION_VERTICAL Spacer should expand vertically.

UGLI_LI_F_SPACER_DIRECTION_BOTH Spacer should expand in both directions.

Function Documentation

UGLILayoutItem NewSpacer (UGLIWindow *window*, SpacerDirectionFlag *flags*)

Constructor that creates a new spacer.

Parameters:

Returns:

window	Window, the LayoutItem will be added to.
flags	flags (currently define only the directions).

new LayoutItem (if successful, otherwise NULL)

---

lightprofile.h File Reference

Light profile.

```
#include "header_begin.h"
#include "header_end.h"
```

Macros

#define **UGLI_MAX_LIGHTS** 8

Max number of light sources.

Typedefs

typedef unsigned int **UGLILightId**
typedef unsigned int **UGLILightComponentId**
typedef struct
UGLILightProfileData * **UGLILightProfile**

Enumerations

enum **UGLILightComponent** { **UGLILightAmbient**, **UGLILightDiffuse**, **UGLILightSpecular**, **UGLILightPosition** }

Functions

UGLILightProfile UGLINewLightProfile (void)
UGLILightProfile UGLICloneLightProfile (const **UGLILightProfile** profile)
void **UGLIDeleteLightProfile** (**UGLILightProfile** profile)
void **UGLILightProfileSetCurrent** (**UGLILightProfile** profile)
void **UGLILightProfileSetLightingEnabled** (**UGLILightProfile** profile, int enabled)
void **UGLILightProfileSetLightEnabled** (**UGLILightProfile** profile, **UGLILightId** light, int enabled)
void **UGLILightProfileSetLightPropertyV** (**UGLILightProfile** profile, **UGLILightId** light, **UGLILightComponentId** component, const float *values)

Detailed Description

Light profile.

Typedef Documentation

typedef unsigned int UGLILightComponentId

Light component ID.

typedef unsigned int UGLILightId

Light ID.

typedef struct UGLILightProfileData* UGLILightProfile

Light profile object.

Enumeration Type Documentation

enum UGLILightComponent

Valid light components that user can set.

Enumerator:

UGLILightAmbient Ambient component of the light.
UGLILightDiffuse Diffuse component of the light.
UGLILightSpecular Specular component of the light.
UGLILightPosition Position of the light.

Function Documentation

UGLILightProfile UGLICloneLightProfile (const UGLILightProfile *profile*)

Creates a clone of an existing GenLightProfile.

Parameters:

Returns:

profile	GenLightProfile to clone.

Clone of given GenLightProfile (or NULL if failed)

void UGLIDeleteLightProfile (UGLILightProfile *profile*)

Destructor that deletes the specified GenLightProfile.

Parameters:

Returns:

profile	GenLightProfile to delete.

void UGLILightProfileSetCurrent (UGLILightProfile *profile*)

Sets current GenLightProfile.

Parameters:

profile	GenLightProfile to use (or NULL for none)

void UGLILightProfileSetLightEnabled (UGLILightProfile *profile*, UGLILightId *light*, int *enabled*)

Sets whether or not specified light should be enabled in the specified GenLightProfile.

Parameters:

profile	GenLightProfile to use.
light	Light to enable / disable.
enabled	Whether or not the specified light should be enabled.

void UGLILightProfileSetLightingEnabled (UGLILightProfile *profile*, int *enabled*)

Sets whether or not lighting should be enabled in the specified GenLightProfile.

Parameters:

profile	GenLightProfile to use.
enabled	Whether or not the lighting should be enabled.

void UGLILightProfileSetLightPropertyV (UGLILightProfile *profile*, UGLILightId *light*, UGLILightComponentId *component*, const float * *values*)

Sets the specified light component for the specified light should be enabled in the specified GenLightProfile.

Parameters:

profile	GenLightProfile to use.

light	Light to set the component for.
component	Which component to set.
values	New values for the specified light component.

UGLILightProfile UGLINewLightProfile (void)

Constructor that creates a new GenLightProfile.

Returns:

new GenLightProfile (or NULL if failed)

---

maincontext.h File Reference

Main context.

```
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct
UGLIMainContextData * **UGLIMainContext**

Functions

UGLIMainContext UGLINewMainContext (int argc, char *argv[])
void **UGLIDeleteMainContext** (**UGLIMainContext** maincontext)

Detailed Description

Main context.
Initializes library for use. Needed as an argument when opening windows.

Typedef Documentation

typedef struct UGLIMainContextData* UGLIMainContext

Main context object. Needed for opening windows.

Function Documentation

void UGLIDeleteMainContext (UGLIMainContext *maincontext*)

Destructor that deletes the specified MainContext instance.

Parameters:

maincontext	MainContext to delete.

UGLIMainContext UGLINewMainContext (int *argc*, char * *argv*[])

Constructor that creates a new MainContext instance.

Parameters:

argc	Argument count given to program.
argv	Arguments given to program.

new UGLIMainContext (or NULL if failed)

---

matrixmath.h File Reference

Various 4x4 Matrix math functions.

```
#include "header_begin.h"
#include "header_end.h"
```

Functions

float **UGLIMatrixDeterminant** (const float *matrix)
void **UGLIMatrixMultiply** (float *result, const float *first, const float *second)
void **UGLIMatrixRotateAxisAngle** (float *matrix, float angle, float x, float y, float z)
void **UGLIMatrixRotateAxisAngleV** (float *matrix, float angle, const float *axis)
void **UGLIMatrixRotateX** (float *matrix, float angle)
void **UGLIMatrixRotateY** (float *matrix, float angle)
void **UGLIMatrixRotateZ** (float *matrix, float angle)
void **UGLIMatrixTranslate** (float *matrix, float x, float y, float z)
void **UGLIMatrixTranslateV** (float *matrix, float *pos)
void **UGLIMatrixTransformVertex** (const float *matrix, float *vertex)
void **UGLIMatrixCopy** (float *dst, const float *src)

Detailed Description

Various 4x4 Matrix math functions.

Function Documentation

void UGLIMatrixCopy (float * *dst*, const float * *src*)

Copies the specified matrix.

Parameters:

Returns:

src	Source matrix.
dst	Destination matrix.

float UGLIMatrixDeterminant (const float * *matrix*)

Calculates the determinant of specified matrix.

Parameters:

matrix	4x4 Matrix to calculate determinant for.

determinant.

void UGLIMatrixMultiply (float * *result*, const float * *first*, const float * *second*)

Multiplies 2 4x4 matrices into third one.

Parameters:

Returns:

result	Result matrix. Must be different from first & second.
first	First matrix.
second	Second matrix.

void UGLIMatrixRotateAxisAngle (float * *matrix*, float *angle*, float *x*, float *y*, float *z*)

Does axis angle rotation for the specified matrix.

Parameters:

matrix	Matrix to rotate.
angle	Angle of rotation.
x	X component of the rotation axis.
y	Y component of the rotation axis.
z	Z component of the rotation axis.

void UGLIMatrixRotateAxisAngleV (float * *matrix*, float *angle*, const float * *axis*)

Does axis angle rotation for the specified matrix.

Parameters:

matrix	Matrix to rotate.
angle	Angle of rotation.
axis	pointer to float[3] that defines the rotation axis.

void UGLIMatrixRotateX (float * *matrix*, float *angle*)

Rotation for the specified matrix along X axis.

Parameters:

matrix	Matrix to rotate.
angle	Angle of rotation.

void UGLIMatrixRotateY (float * *matrix*, float *angle*)

Rotation for the specified matrix along Y axis.

Parameters:

matrix	Matrix to rotate.
angle	Angle of rotation.

void UGLIMatrixRotateZ (float * *matrix*, float *angle*)

Rotation for the specified matrix along Z axis.

Parameters:

matrix	Matrix to rotate.
angle	Angle of rotation.

void UGLIMatrixTransformVertex (const float * *matrix*, float * *vertex*)

Transforms the specified 3D vertex by the given matrix.

Parameters:

matrix	Matrix to translate with.
vertex	Vertex to transform.

void UGLIMatrixTranslate (float * *matrix*, float *x*, float *y*, float *z*)

Translates the specified matrix.

Parameters:

matrix	Matrix to translate.
x	X translation..
y	Y translation.
z	Z translation.

void UGLIMatrixTranslateV (float * *matrix*, float * *pos*)

Translates the specified matrix.

Parameters:

matrix	Matrix to translate.
pos	Pointer to float[3] that defines the X/Y/Z translations.

---

matrixmathinit.h File Reference

Various 4x4 Matrix initialization functions.

```
#include <string.h>
#include "matrixmath.h"
#include "header_begin.h"
#include "header_end.h"
```

Functions

void **UGLIMatrixInitIdentity** (float *matrix)
void **UGLIMatrixInitRotateAxisAngle** (float *matrix, float angle, float x, float y, float z)
void **UGLIMatrixInitRotateX** (float *matrix, float angle)
void **UGLIMatrixInitRotateY** (float *matrix, float angle)
void **UGLIMatrixInitRotateZ** (float *matrix, float angle)
void **UGLIMatrixInitTranslate** (float *matrix, float x, float y, float z)

Detailed Description

Various 4x4 Matrix initialization functions.

Function Documentation

void UGLIMatrixInitIdentity (float * *matrix*)

Sets the specified matrix to identity.

Parameters:

matrix	4x4 Matrix to set to identity.

void UGLIMatrixInitRotateAxisAngle (float * *matrix*, float *angle*, float *x*, float *y*, float *z*)

Sets the specified matrix to the specified axis angle rotation.

Parameters:

matrix	Matrix to set.
angle	Angle of rotation.
x	X component of the rotation axis.
y	Y component of the rotation axis.
z	Z component of the rotation axis.

void UGLIMatrixInitRotateX (float * *matrix*, float *angle*)

Sets the specified matrix to the specified X axis rotation.

Parameters:

matrix	Matrix to set.
angle	Angle of rotation.

void UGLIMatrixInitRotateY (float * *matrix*, float *angle*)

Sets the specified matrix to the specified Y axis rotation.

Parameters:

matrix	Matrix to set.
angle	Angle of rotation.

void UGLIMatrixInitRotateZ (float * *matrix*, float *angle*)

Sets the specified matrix to the specified Z axis rotation.

Parameters:

matrix	Matrix to set.
angle	Angle of rotation.

void UGLIMatrixInitTranslate (float * *matrix*, float *x*, float *y*, float *z*)

Sets the specified matrix to the specified translation.

Parameters:

matrix	Matrix to set.
x	X translation.
y	Y translation.
z	Z translation.

---

matrixprojection.h File Reference

Matrix projection initialization functions.

```
#include "matrixmath.h"
#include "header_begin.h"
#include "header_end.h"
```

Functions

void **UGLIMatrixInitFrustum** (float *matrix, float left, float right, float bottom, float top, float near, float far)

void **UGLIMatrixInitPerspective** (float *matrix, float fovy, float aspect, float near, float far)

void **UGLIMatrixInitOrtho** (float *matrix, float left, float right, float bottom, float top, float near, float far)

int **UGLIMatrixProject3DTo2D** (const float *pos, const float *modelview, const float *projection, const int *viewport, float *windowcoord)

Detailed Description

Matrix projection initialization functions.

Function Documentation

void UGLIMatrixInitFrustum (float * *matrix*, float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Initializes the specified matrix so that it matches the specified frustum transformation.

Parameters:

matrix	Matrix to initialize.
left	Left plane.
right	Right plane.
bottom	Bottom plane.
top	Top plane.
near	Near plane.
far	Far plane.

void UGLIMatrixInitOrtho (float * *matrix*, float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Initializes the specified matrix so that it matches the specified orthogonal transformation.

Parameters:

matrix	Matrix to initialize.
left	Left plane.
right	Right plane.
bottom	Bottom plane.
top	Top plane.
near	Near plane.
far	Far plane.

void UGLIMatrixInitPerspective (float * *matrix*, float *fovy*, float *aspect*, float *near*, float *far*)

Initializes the specified matrix so that it matches the specified perspective transformation.

Parameters:

matrix	Matrix to initialize.
fovy	Y Field-of-vision.
aspect	Aspect ratio.
near	Near plane.
far	Far plane.

int UGLIMatrixProject3DTo2D (const float * *pos*, const float * *modelview*, const float * *projection*, const int * *viewport*, float * *windowcoord*)

Projects 3D coordinates to 2D.

Parameters:

pos	3D coordinates.
modelview	Modelview matrix.
projection	Projection matrix.
viewport	Current viewport.

windowcoord	Array for resulting window coordinates.

0 on success.

---

matrixstack.h File Reference

Matrix stack & related mathematics.

```
#include "matrixmath.h"
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct
UGLIMatrixStackData * **UGLIMatrixStack**

Enumerations

enum **UGLIMatrixStackError** { **UGLI_MS_NO_ERROR** = 0,
UGLI_MS_INVALID_MATRIX_STACK,
UGLI_MS_INVALID_SOURCE_POINTER,
UGLI_MS_INVALID_TARGET_POINTER,
UGLI_MS_STACK_OVERFLOW, **UGLI_MS_STACK_UNDERFLOW**,
UGLI_MS_ERROR_VALUE_NOT_IMPLEMENTED }

Functions

UGLIMatrixStack UGLINewMatrixStack (unsigned int depth)
void **UGLIDeleteMatrixStack** (**UGLIMatrixStack** stack)
UGLIMatrixStackError UGLIMatrixStackPushMatrix (**UGLIMatrixStack** stack)
UGLIMatrixStackError UGLIMatrixStackPopMatrix (**UGLIMatrixStack** stack)
UGLIMatrixStackError UGLIMatrixStackSetIdentity (**UGLIMatrixStack** stack)
UGLIMatrixStackError UGLIMatrixStackLoadMatrix (**UGLIMatrixStack** stack, const float *matrix)
UGLIMatrixStackError UGLIMatrixStackMultMatrix (**UGLIMatrixStack** stack, const float *matrix)
UGLIMatrixStackError UGLIMatrixStackPreMultMatrix (**UGLIMatrixStack** stack, const float *matrix)
UGLIMatrixStackError UGLIMatrixStackTranslate (**UGLIMatrixStack** stack, float x, float y, float z)
UGLIMatrixStackError UGLIMatrixStackScale (**UGLIMatrixStack** stack, float x, float y, float z)
UGLIMatrixStackError UGLIMatrixStackSetScale (**UGLIMatrixStack** stack, float x, float y, float z)
UGLIMatrixStackError UGLIMatrixStackRotateAxis (**UGLIMatrixStack** stack, float angle, float x, float y, float z)
UGLIMatrixStackError UGLIMatrixStackRotateAxisV (**UGLIMatrixStack** stack, float angle, const float *axis)

UGLIMatrixStackError UGLIMatrixStackRotateX (**UGLIMatrixStack** stack, float angle)
UGLIMatrixStackError UGLIMatrixStackRotateY (**UGLIMatrixStack** stack, float angle)
UGLIMatrixStackError UGLIMatrixStackRotateZ (**UGLIMatrixStack** stack, float angle)
UGLIMatrixStackError UGLIMatrixStackMultFrustum (**UGLIMatrixStack** stack, float left, float right, float bottom, float top, float near, float far)
UGLIMatrixStackError UGLIMatrixStackMultPerspective (**UGLIMatrixStack** stack, float fovy, float aspect, float near, float far)
UGLIMatrixStackError UGLIMatrixStackMultOrtho (**UGLIMatrixStack** stack, float left, float right, float bottom, float top, float near, float far)
UGLIMatrixStackError UGLIMatrixStackInitFrustum (**UGLIMatrixStack** stack, float left, float right, float bottom, float top, float near, float far)
UGLIMatrixStackError UGLIMatrixStackInitPerspective (**UGLIMatrixStack** stack, float fovy, float aspect, float near, float far)
UGLIMatrixStackError UGLIMatrixStackInitOrtho (**UGLIMatrixStack** stack, float left, float right, float bottom, float top, float near, float far)
UGLIMatrixStackError UGLIMatrixStackProject3DTo2D (const float *pos, const **UGLIMatrixStack** modelview, const **UGLIMatrixStack** projection, const int *viewport, float *windowcoord)
UGLIMatrixStackError UGLIMatrixStackGetMatrix (const **UGLIMatrixStack** stack, float *matrix)
const float * **UGLIMatrixStackGetMatrixPointer** (const **UGLIMatrixStack** stack)
UGLIMatrixStackError UGLIMatrixStackTransformVertex (const **UGLIMatrixStack** stack, float *vertex)
UGLIMatrixStackError UGLIMatrixStackTransformMatrix (const **UGLIMatrixStack** stack, float *matrix)

Detailed Description

Matrix stack & related mathematics.

Typedef Documentation

typedef struct UGLIMatrixStackData* UGLIMatrixStack

Matrix stack object.

Enumeration Type Documentation

enum UGLIMatrixStackError

Possible error values for various error states. No error should be considered "unimportant warning" Return value on success is always MS_NO_ERROR (0).

Enumerator:

UGLI_MS_NO_ERROR No error.
UGLI_MS_INVALID_MATRIX_STACK Matrix stack pointer == NULL.
UGLI_MS_INVALID_SOURCE_POINTER Source pointer == NULL.
UGLI_MS_INVALID_TARGET_POINTER Target pointer == NULL.
UGLI_MS_STACK_OVERFLOW Stack overflow (trying to push over depth).
UGLI_MS_STACK_UNDERFLOW Stack underflow (trying to pop under 0).
UGLI_MS_ERROR_VALUE_NOT_IMPLEMENTED Currently unimplemented error value.

Function Documentation

void UGLIDeleteMatrixStack (UGLIMatrixStack *stack*)

Destructor. Deletes specified UGLIMatrixStack.

Parameters:

Returns:

stack	Matrix stack to delete.

UGLIMatrixStackError UGLIMatrixStackGetMatrix (const UGLIMatrixStack *stack*, float * *matrix*)

Gets a copy of current matrix from the specified matrix stack.

Parameters:

stack	Stack to get matrix from.
matrix	Pointer where to store the matrix.

error id.

const float* UGLIMatrixStackGetMatrixPointer (const UGLIMatrixStack *stack*)

Gets a const pointer to the current matrix in the specified stack.

Parameters:

Returns:

stack	Stack to get matrix from.

const pointer to the current matrix in the stack.

UGLIMatrixStackError UGLIMatrixStackInitFrustum (UGLIMatrixStack *stack*, float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Inits current matrix in stack so that it matches the specified frustum.

Parameters:

Returns:

stack	Matrix stack to do operation in.
left	Left plane.
right	Right plane.
bottom	Bottom plane.
top	Top plane.
near	Near plane.
far	Far plane.

error id.

UGLIMatrixStackError UGLIMatrixStackInitOrtho (UGLIMatrixStack *stack*, float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Inits current matrix in stack so that it matches the specified ortho projection.

Parameters:

Returns:

stack	Matrix stack to do operation in.
left	Left plane.
right	Right plane.
bottom	Bottom plane.
top	Top plane.
near	Near plane.
far	Far plane.

error id.

UGLIMatrixStackError UGLIMatrixStackInitPerspective (UGLIMatrixStack *stack*, float *fovy*, float *aspect*, float *near*, float *far*)

Inits current matrix in stack so that it matches the specified perspective.

Parameters:

Returns:

stack	Matrix stack to do operation in.
fovy	Y Field-of-vision.
aspect	Aspect ratio.
near	Near plane.
far	Far plane.

error id.

UGLIMatrixStackError UGLIMatrixStackLoadMatrix (UGLIMatrixStack *stack*, const float * *matrix*)

Loads the specified matrix to current position of the specified MatrixStack.

Parameters:

Returns:

stack	Matrix stack to do operation in.
matrix	Matrix to load.

error id.

UGLIMatrixStackError UGLIMatrixStackMultFrustum (UGLIMatrixStack *stack*, float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Multiplies current matrix in stack by matrix that matches the specified frustum.

Parameters:

Returns:

stack	Matrix stack to do operation in.
left	Left plane.
right	Right plane.
bottom	Bottom plane.
top	Top plane.
near	Near plane.
far	Far plane.

error id.

UGLIMatrixStackError UGLIMatrixStackMultMatrix (UGLIMatrixStack *stack*, const float * *matrix*)

Multiplies the top matrix in the specified MatrixStack with the specified matrix.

Parameters:

Returns:

stack	Matrix stack to do operation in.
matrix	Matrix to multiply with.

error id.

UGLIMatrixStackError UGLIMatrixStackMultOrtho (UGLIMatrixStack *stack*, float *left*, float *right*, float *bottom*, float *top*, float *near*, float *far*)

Multiplies current matrix in stack by matrix that matches the specified ortho projection.

Parameters:

Returns:

stack	Matrix stack to do operation in.
left	Left plane.
right	Right plane.
bottom	Bottom plane.

top	Top plane.
near	Near plane.
far	Far plane.

error id.

UGLIMatrixStackError UGLIMatrixStackMultPerspective (UGLIMatrixStack *stack*, float *fovy*, float *aspect*, float *near*, float *far*)

Multiplies current matrix in stack by matrix that matches the specified projection.

Parameters:

Returns:

stack	Matrix stack to do operation in.
fovy	Y Field-of-vision.
aspect	Aspect ratio.
near	Near plane.
far	Far plane.

error id.

UGLIMatrixStackError UGLIMatrixStackPopMatrix (UGLIMatrixStack *stack*)

Pop matrix.

Parameters:

Returns:

stack	Matrix stack to do operation in.

error id.

UGLIMatrixStackError UGLIMatrixStackPreMultMatrix (UGLIMatrixStack *stack*, const float * *matrix*)

Pre-multiplies the top matrix in the specified MatrixStack with the specified matrix.

Parameters:

Returns:

stack	Matrix stack to do operation in.
matrix	Matrix to premultiply with.

error id.

UGLIMatrixStackError UGLIMatrixStackProject3DTo2D (const float * *pos*, const UGLIMatrixStack *modelview*, const UGLIMatrixStack *projection*, const int * *viewport*, float * *windowcoord*)

Projects 3D coordinates to 2D.

Parameters:

Returns:

pos	3D coordinates.
modelview	Modelview matrix stack.
projection	Projection matrix stack.
viewport	Current viewport.

windowcoord	Array for resulting window coordinates.

error id.

UGLIMatrixStackError UGLIMatrixStackPushMatrix (UGLIMatrixStack *stack*)

Push matrix.

Parameters:

Returns:

stack	Matrix stack to do operation in.

error id.

UGLIMatrixStackError UGLIMatrixStackRotateAxis (UGLIMatrixStack *stack*, float *angle*, float *x*, float *y*, float *z*)

Multiplies the current position of the specified MatrixStack with axis angle rotation.

Parameters:

Returns:

stack	Matrix stack to do operation in.
angle	Rotation angle.
x	X component of rotation axis.
y	Y component of rotation axis.
z	Z component of rotation axis.

error id.

UGLIMatrixStackError UGLIMatrixStackRotateAxisV (UGLIMatrixStack *stack*, float *angle*, const float * *axis*)

Multiplies the current position of the specified MatrixStack with axis angle rotation.

Parameters:

Returns:

stack	Matrix stack to do operation in.
angle	Rotation angle.
axis	Pointer to float[3] that specifies the rotation axis.

error id.

UGLIMatrixStackError UGLIMatrixStackRotateX (UGLIMatrixStack *stack*, float *angle*)

Multiplies the current position of the specified MatrixStack with specified rotation along X axis.

Parameters:

Returns:

stack	Matrix stack to do operation in.
angle	Rotation angle.

error id.

UGLIMatrixStackError UGLIMatrixStackRotateY (UGLIMatrixStack *stack*, float *angle*)

Multiplies the current position of the specified MatrixStack with specified rotation along Y axis.

Parameters:

Returns:

stack	Matrix stack to do operation in.
angle	Rotation angle.

error id.

UGLIMatrixStackError UGLIMatrixStackRotateZ (UGLIMatrixStack *stack*, float *angle*)

Multiplies the current position of the specified MatrixStack with specified rotation along Z axis.

Parameters:

Returns:

stack	Matrix stack to do operation in.
angle	Rotation angle.

error id.

UGLIMatrixStackError UGLIMatrixStackScale (UGLIMatrixStack *stack*, float *x*, float *y*, float *z*)

Scales the top matrix in the specified MatrixStack with the specified values.

Parameters:

Returns:

stack	Matrix stack to do operation in.
x	X scale.
y	Y scale.
z	Z scale.

error id.

UGLIMatrixStackError UGLIMatrixStackSetIdentity (UGLIMatrixStack *stack*)

Sets current matrix in stack to identity.

Parameters:

Returns:

stack	Matrix stack to do operation in.

error id.

UGLIMatrixStackError UGLIMatrixStackSetScale (UGLIMatrixStack *stack*, float *x*, float *y*, float *z*)

Sets the scale of the top matrix in the specified MatrixStack (replacing old values).

Parameters:

Returns:

stack	Matrix stack to do operation in.
x	New x scale.
y	New y scale.
z	New z scale.

error id.

UGLIMatrixStackError UGLIMatrixStackTransformMatrix (const UGLIMatrixStack *stack*, float * *matrix*)

Transforms the specified matrix with the current matrix from the specified matrix stack.

Parameters:

Returns:

stack	Stack to get matrix from.
matrix	Matrix to transform.

Error id.

UGLIMatrixStackError UGLIMatrixStackTransformVertex (const UGLIMatrixStack *stack*, float * *vertex*)

Transforms the specified vertex with the current matrix from the specified matrix stack.

Parameters:

Returns:

stack	Stack to get matrix from.
vertex	Vertex to transform.

Error id.

UGLIMatrixStackError UGLIMatrixStackTranslate (UGLIMatrixStack *stack*, float *x*, float *y*, float *z*)

Translates the top matrix in the specified MatrixStack with the specified values.

Parameters:

Returns:

stack	Matrix stack to do operation in.
x	X translation.
y	Y translation.
z	Z translation.

error id.

UGLIMatrixStack UGLINewMatrixStack (unsigned int *depth*)

Constructor. Creates new UGLIMatrixStack.

Parameters:

Returns:

depth	Matrix stack depth.

New UGLIMatrixStack.

---

memory.h File Reference

Low level memory management.

```
#include <stdlib.h>
#include "memory_p.h"
#include "header_begin.h"
#include "header_end.h"
```

Macros

#define **UGLIRealloc**(ptr, size) UGLIRealloc_fast(ptr, size)
#define **UGLIMalloc**(size) UGLIMalloc_fast(size)
#define **UGLIFree**(ptr) UGLIFree_fast(ptr)

Detailed Description

Low level memory management.

Macro Definition Documentation

#define UGLIFree(ptr) UGLIFree_fast(ptr)

Frees specified memory allocation.

Parameters:

Returns:

ptr	Pointer to free.

#define UGLIMalloc(size) UGLIMalloc_fast(size)

Allocates specified amount of memory.

Parameters:

size	allocation size.

New pointer (NULL if failed).

#define UGLIRealloc(ptr, size) UGLIRealloc_fast(ptr, size)

Tries to change specified allocation size to the one specified.

Parameters:

Returns:

ptr	Pointer to reallocate. If NULL, acts like UGLIMalloc.
size	New allocation size.

New pointer (NULL if failed, or if size==0).

---

minmax.h File Reference

MIN & MAX macros.

```
#include "header_begin.h"
#include "header_end.h"
```

Macros

#define **MIN**(first, second) ((first<second) ? (first) : (second))
#define **MAX**(first, second) ((first>second) ? (first) : (second))

Detailed Description

MIN & MAX macros.

Macro Definition Documentation

#define MAX(first, second) ((first>second) ? (first) : (second))

Returns the biggest of the 2 given values.

Parameters:

Returns:

first	First value.
second	Second value.

Biggest value.

#define MIN(first, second) ((first<second) ? (first) : (second))

Returns the smallest of the 2 given values.

Parameters:

Returns:

first	First value.
second	Second value.

Smallest value.

---

mutex.h File Reference

Mutex object.

```
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct UGLIMutexData * **UGLIMutex**

Functions

UGLIMutex UGLINewMutex (void)
void **UGLIDeleteMutex** (**UGLIMutex** mutex)
int **UGLIMutexLock** (**UGLIMutex** mutex)
int **UGLIMutexUnlock** (**UGLIMutex** mutex)

Detailed Description

Mutex object.

Typedef Documentation

typedef struct UGLIMutexData* UGLIMutex

GenMutex object.

Function Documentation

void UGLIDeleteMutex (UGLIMutex *mutex*)

Destructor. Deletes the specified Mutex object.

Parameters:

Returns:

mutex	Mutex to delete.

int UGLIMutexLock (UGLIMutex *mutex*)

Locks specified Mutex.

Parameters:

mutex	Mutex to lock.

0 on success.

int UGLIMutexUnlock (UGLIMutex *mutex*)

Unlocks specified Mutex.

Parameters:

Returns:

mutex	Mutex to unlock.

0 on success.

UGLIMutex UGLINewMutex (void)

Constructor. Creates new Mutex object.

Returns:

New Mutex object.

---

normalize.h File Reference

Vertex Array Normalizer.

```
#include "vertexarray.h"
#include "header_begin.h"
#include "header_end.h"
```

Macros

#define **UGLI_NORMALIZE_MERGE_VERTICES** 0x01
#define **UGLI_NORMALIZE_INDEX_VERTICES** 0x02
#define **UGLI_NORMALIZE_PROPOTIONAL_TO_ANGLE** 0x04

Enumerations

enum **UGLIShadeModel** { **UGLI_SHADEMODEL_NONE** = 0,
UGLI_SHADEMODEL_SMOOTH, **UGLI_SHADEMODEL_FLAT** }

Functions

void **UGLIVertexArrayNormalize** (**UGLIVertexArray** vertexarray, unsigned int settings, **UGLIShadeModel** mode, float anglelimit, float mergethreshold)

Detailed Description

Vertex Array Normalizer.

Macro Definition Documentation

#define UGLI_NORMALIZE_INDEX_VERTICES 0x02

Create indexes for Mesh.

#define UGLI_NORMALIZE_MERGE_VERTICES 0x01

Merge vertices that are close enough to each other (fixes shading).

#define UGLI_NORMALIZE_PROPOTIONAL_TO_ANGLE 0x04

Give more weight for faces, for which vertex has a bigger angle.

Enumeration Type Documentation

enum UGLIShadeModel

Valid normalization modes.

Enumerator:

UGLI_SHADEMODEL_NONE No shade model specified.
UGLI_SHADEMODEL_SMOOTH Smooth shading. Edges should be rounded.
UGLI_SHADEMODEL_FLAT Flat shading. No edges should be rounded, and all triangles should appear perfectly flat, each vertex using same (face) normal.

Function Documentation

void UGLIVertexArrayNormalize (UGLIVertexArray *vertexarray*, unsigned int *settings*, UGLIShadeModel *mode*, float *anglelimit*, float *mergethreshold*)

Normalizes specified Mesh.

Parameters:

Returns:

vertexarray	Vertex array to normalize.
settings	Normalization-related settings.
mode	Shading mode.
anglelimit	Angle limit for smooth shading.
mergethreshold	Merging threshold for combining vertices, if specified.

---

particles.h File Reference

Particle drawing.

```
#include "texture.h"
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct
UGLIParticleGroupData * **UGLIParticleGroup**

Functions

UGLIParticleGroup UGLINewParticleGroup (**UGLITexture** texture, float radius, unsigned int size)
void **UGLIDeleteParticleGroup** (**UGLIParticleGroup** pg)
void **UGLIParticleGroupStartDraw** (**UGLIParticleGroup** pg)
void **UGLIParticleGroupAddParticle3fv** (**UGLIParticleGroup** pg, const float *pos)
void **UGLIParticleGroupAddParticle3f** (**UGLIParticleGroup** pg, float x, float y, float z)
void **UGLIParticleGroupAddParticle2f** (**UGLIParticleGroup** pg, float x, float y)
void **UGLIParticleGroupEndDraw** (**UGLIParticleGroup** pg)

Detailed Description

Particle drawing.

Typedef Documentation

typedef struct UGLIParticleGroupData* UGLIParticleGroup

UGLIParticleGroup object.

Function Documentation

void UGLIDeleteParticleGroup (UGLIParticleGroup *pg*)

Destructor that deletes the specified UGLIParticleGroup.

Parameters:

pg	UGLIParticleGroup to delete.

UGLIParticleGroup UGLINewParticleGroup (UGLITexture *texture*, float *radius*, unsigned int *size*)

Constructor that creates a new UGLIParticleGroup.

Parameters:

texture	Texture to use for particles.
radius	Radius of a single particle.
size	Size of particle group. Can be smaller than amount of particles you will draw at once.

new UGLIParticleGroup.

void UGLIParticleGroupAddParticle2f (UGLIParticleGroup *pg*, float *x*, float *y*)

Draws a single 2D (Z=0) particle with the specified UGLIParticleGroup.

Parameters:

Returns:

pg	UGLIParticleGroup to draw with.
x	Particle X position.
y	Particle Y position.

void UGLIParticleGroupAddParticle3f (UGLIParticleGroup *pg*, float *x*, float *y*, float *z*)

Draws a single 3D particle with the specified UGLIParticleGroup.

Parameters:

pg	UGLIParticleGroup to draw with.
x	Particle X position.
y	Particle Y position.
z	Particle Z position.

void UGLIParticleGroupAddParticle3fv (UGLIParticleGroup *pg*, const float * *pos*)

Draws a single 3D particle with the specified UGLIParticleGroup.

Parameters:

pg	UGLIParticleGroup to draw with.
pos	Pointer to float[3] that specifies the particle position.

void UGLIParticleGroupEndDraw (UGLIParticleGroup *pg*)

Ends particle drawing with the specified UGLIParticleGroup.

Parameters:

pg	UGLIParticleGroup to end drawing with.

void UGLIParticleGroupStartDraw (UGLIParticleGroup *pg*)

Start drawing particles with the specified UGLIParticleGroup.

Parameters:

pg	UGLIParticleGroup to draw with.

---

physics.h File Reference

Physics handler.

```
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct UGLIPhysicsData * **UGLIPhysics**

Functions

UGLIPhysics UGLINewPhysicsHandler (void(*VariableDeltaPhysics)(void *data, unsigned int delta), void(*ConstantDeltaPhysics)(void *data), void *data, unsigned int delta)
void **UGLIDeletePhysicsHandler** (**UGLIPhysics** physics)
void **UGLIResetPhysics** (**UGLIPhysics** physics)
void **UGLIRunPhysics** (**UGLIPhysics** physics)

Detailed Description

Physics handler.

Typedef Documentation

typedef struct UGLIPhysicsData* UGLIPhysics

Physics object.

Function Documentation

void UGLIDeletePhysicsHandler (UGLIPhysics *physics*)

Destructor that deletes the specified handler.

Parameters:

physics	handler to delete.

UGLIPhysics UGLINewPhysicsHandler (void(*)(void *data, unsigned int delta) *VariableDeltaPhysics*, void(*)(void *data) *ConstantDeltaPhysics*, void * *data*, unsigned int *delta*)

Constructor that creates a new handler.

Parameters:

VariableDeltaPhysics	Function that can be called with variable delta.
ConstantDeltaPhysics	Function that needs to be called with constant delta.
data	Data pointer.
delta	Delta to use for ConstantDeltaPhysics.

new handler.

void UGLIResetPhysics (UGLIPhysics *physics*)

Resets physics. Sets delta to zero.

Parameters:

Returns:

physics	to reset.

void UGLIRunPhysics (UGLIPhysics *physics*)

Runs the specified physics handler once. Runs the specified constant & variable delta physics.

Parameters:

physics	Physics handler to run.

---

pointerarray.h File Reference

pointer array.

```
#include <stdlib.h>
#include "header_begin.h"
#include "header_end.h"
```

Data Structures

struct **UGLIPointerArray**
Pointer array. **Typedefs**

typedef struct **UGLIPointerArray UGLIPointerArray**
Pointer array.

Functions

void **UGLIInitPointerArray** (**UGLIPointerArray** *pointerarray)
UGLIPointerArray * UGLINewPointerArray (void)
size_t **UGLIPointerArrayAppendPointer** (**UGLIPointerArray** *pointerarray, const void *const element)
size_t **UGLIPointerArrayHasSpace** (**UGLIPointerArray** *pointerarray, size_t elementsrequired)
void * **UGLIPointerArrayGetPointer** (const **UGLIPointerArray** *pointerarray, size_t item)

Detailed Description

pointer array.

Function Documentation

void UGLIInitPointerArray (UGLIPointerArray * *pointerarray*)

Initializes the specified pointer array.

Parameters:

pointerarray	Pointer array to initialize.

UGLIPointerArray* UGLINewPointerArray (void)

Constructor that creates a new pointer array.

Returns:

New pointer array (if successful, otherwise NULL).

size_t UGLIPointerArrayAppendPointer (UGLIPointerArray * *pointerarray*, const void *const *element*)

Appends pointer to the specified pointer array.

Parameters:

pointerarray	Pointer array to append to.
element	Element to append.

Array size.

void* UGLIPointerArrayGetPointer (const UGLIPointerArray * *pointerarray*, size_t *item*)

Gets the specified pointer from the specified pointer array.

Parameters:

Returns:

pointerarray	Pointer array to get pointer from.
item	Which element to get.

pointer.

size_t UGLIPointerArrayHasSpace (UGLIPointerArray * *pointerarray*, size_t *elementsrequired*)

Makes sure the specified pointer array has enough space for specified amount of elements.

Parameters:

Returns:

pointerarray	Pointer array to check.
elementsrequired	How much space is needed.

Array size.

---

quat.h File Reference

Quaternion element.

```
#include "header_begin.h"
#include <string.h>
#include "header_end.h"
```

Data Structures

struct **UGLIQuat**

Functions

UGLIQuat * UGLINewQuat (float x, float y, float z, float angle)
void **UGLIInitQuat** (**UGLIQuat** *quaternion, float x, float y, float z, float angle)
void **UGLICopyQuat** (**UGLIQuat** *copy, const **UGLIQuat** *original)

Creates a copy of the specified quaternion.

void **UGLIDeleteQuat** (**UGLIQuat** *quaternion)
void **UGLIQuatNormalize** (**UGLIQuat** *quaternion)
void **UGLIQuatAddQuat** (**UGLIQuat** *first, const **UGLIQuat** *second)
first = first + second
void **UGLIQuatSubQuat** (**UGLIQuat** *first, const **UGLIQuat** *second)
first = first - second
void **UGLIQuatSubQuatToResult** (**UGLIQuat** *result, const **UGLIQuat** *first, const **UGLIQuat** *second)
result = first - second
void **UGLIQuatMultQuat** (**UGLIQuat** *result, const **UGLIQuat** *first, const **UGLIQuat** *second)
*result = first * second*
void **UGLIQuatMultScalar** (**UGLIQuat** *quaternion, float scalar)
*quaternion = quaternion * scalar*
void **UGLIQuatMultScalarToResult** (**UGLIQuat** *result, const **UGLIQuat** *quaternion, float scalar)
*result = quaternion * scalar*
void **UGLIQuatToMatrix** (float *matrix, const **UGLIQuat** *quaternion)
matrix = matrix(quaternion)
float **UGLIQuatDotProduct** (const **UGLIQuat** *first, const **UGLIQuat** *second)
Calculates quaternion dot product.

Detailed Description

Quaternion element.

Function Documentation

void UGLICopyQuat (UGLIQuat * *copy*, const UGLIQuat * *original*)

Creates a copy of the specified quaternion.

Creates a copy of the specified quaternion.

Parameters:

Returns:

copy	Quaternion to copy to.
original	Quaternion to copy.

void UGLIDeleteQuat (UGLIQuat * *quaternion*)

Deletes the specified Quaternion.

Parameters:

quaternion	Quaternion to delete.

void UGLIInitQuat (UGLIQuat * *quaternion*, float *x*, float *y*, float *z*, float *angle*)

Initializes the specified Quaternion element.

Parameters:

quaternion	Quaternion to init.
x	X component of rotation vector.
y	Y component of rotation vector.
z	Z component of rotation vector.
angle	Rotation angle.

UGLIQuat* UGLINewQuat (float *x*, float *y*, float *z*, float *angle*)

Constructor that creates a new Quaternion element.

Parameters:

x	X component of rotation vector.
y	Y component of rotation vector.
z	Z component of rotation vector.
angle	Rotation angle.

new Quaternion element.

void UGLIQuatAddQuat (UGLIQuat * *first*, const UGLIQuat * *second*)

first = first + second

Adds the second Quaternion to the first.

Parameters:

Returns:

first	First quaternion.
second	Second quaternion.

float UGLIQuatDotProduct (const UGLIQuat * *first*, const UGLIQuat * *second*)

Calculates quaternion dot product.

Calculates the dot product for 2 given quaternions.

Parameters:

first	First quaternion.
second	Second quaternion.

dot product.

void UGLIQuatMultQuat (UGLIQuat * *result*, const UGLIQuat * *first*, const UGLIQuat * *second*)

result = first * second

Multiplies the first quaternion by second to result.

Parameters:

Returns:

result	Result quaternion.
first	First quaternion.
second	Second quaternion.

void UGLIQuatMultScalar (UGLIQuat * *quaternion*, float *scalar*)

quaternion = quaternion * scalar

Multiplies quaternion by the given scalar value.

Parameters:

quaternion	Quaternion.
scalar	Scalar value.

void UGLIQuatMultScalarToResult (UGLIQuat * *result*, const UGLIQuat * *quaternion*, float *scalar*)

result = quaternion * scalar

Multiplies quaternion by the given scalar value to result.

Parameters:

result	Result quaternion.
quaternion	Quaternion.
scalar	Scalar value.

void UGLIQuatNormalize (UGLIQuat * *quaternion*)

normalizes the specified Quaternion.

Parameters:

quaternion	Quaternion to normalize.

void UGLIQuatSubQuat (UGLIQuat * *first*, const UGLIQuat * *second*)

first = first - second

Substracts the second Quaternion to the first.

Parameters:

first	First quaternion.
second	Second quaternion.

void UGLIQuatSubQuatToResult (UGLIQuat * *result*, const UGLIQuat * *first*, const UGLIQuat * *second*)

result = first - second

Substracts the second Quaternion to the first to result.

Parameters:

result	Result quaternion.
first	First quaternion.
second	Second quaternion.

void UGLIQuatToMatrix (float * *matrix*, const UGLIQuat * *quaternion*)

matrix = matrix(quaternion)

Converts the specified quaternion to a 4*4 matrix.

Parameters:

matrix	4*4 matrix.
quaternion	Quaternion to convert.

---

queue.h File Reference

Queue element.

```
#include <stdlib.h>
#include "memory.h"
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct UGLIQueueData * **UGLIQueue**

Functions

UGLIQueue UGLINewQueue (void)
void **UGLIDeleteQueue** (**UGLIQueue** queue)
int **UGLIQueueAppendElement** (**UGLIQueue** queue, void *data)
void * **UGLIQueueRemoveElement** (**UGLIQueue** queue)

Detailed Description

Queue element.

Typedef Documentation

typedef struct UGLIQueueData* UGLIQueue

Queue is a simple datatype, to which you add and remove elements. Works as a FIFO buffer.Queue object.

Function Documentation

void UGLIDeleteQueue (UGLIQueue *queue*)

Destructor that deletes the specified queue element.

Parameters:

queue	Queue to delete.

UGLIQueue UGLINewQueue (void)

Constructor that creates a FIFO queue element.

Returns:

new Queue element

int UGLIQueueAppendElement (UGLIQueue *queue*, void * *data*)

Append element to Queue.

Parameters:

queue	Queue to append element to.
data	Element to append.

0 on success.

void* UGLIQueueRemoveElement (UGLIQueue *queue*)

Removes element from Queue.

Parameters:

Returns:

queue	Queue to remove element from.

element (or NULL if empty).

---

reallocarray.h File Reference

realloc array.

```
#include <stdlib.h>
```

```
#include "header_begin.h"
#include "header_end.h"
```

Data Structures

struct **UGLIReallocArray**

Realloc array. **Typedefs**

typedef struct **UGLIReallocArray UGLIReallocArray**

Realloc array.

Functions

void **UGLIInitReallocArray** (**UGLIReallocArray** *reallocarray, size_t elementsize)
UGLIReallocArray * **UGLINewReallocArray** (size_t elementsize)
void **UGLICleanReallocArray** (**UGLIReallocArray** *reallocarray)
void **UGLIDeleteReallocArray** (**UGLIReallocArray** *reallocarray)
size_t **UGLIReallocArrayHasSpace** (**UGLIReallocArray** *reallocarray, size_t elementsrequired)
size_t **UGLIReallocArraySetSize** (**UGLIReallocArray** *reallocarray, size_t size)
int **UGLIReallocArrayFinalize** (**UGLIReallocArray** *reallocarray)
void * **UGLIReallocArrayGetElement** (const **UGLIReallocArray** *reallocarray, size_t item)
void **UGLIReallocArraySetElement** (**UGLIReallocArray** *reallocarray, size_t item, const void *const element)

Sets the specified ReallocArray element.

size_t **UGLIReallocArrayAppendElement** (**UGLIReallocArray** *reallocarray, const void *const element)
size_t **UGLIReallocArrayAppendPartialElement** (**UGLIReallocArray** *reallocarray, const void *const element, size_t bytes)

Appends the specified partial element to ReallocArray.

void **UGLIReallocArraySetArray** (**UGLIReallocArray** *reallocarray, void *array, size_t elements)
void **UGLIReallocArrayRemoveElementById** (**UGLIReallocArray** *reallocarray, size_t elementid)
void **UGLIReallocArrayRemoveElementsById** (**UGLIReallocArray** *reallocarray, size_t elementid, size_t amount)

Detailed Description

realloc array.

Typedef Documentation

typedef struct UGLIReallocArray UGLIReallocArray

Realloc array.

ReallocTable is a container, where all elements are guaranteed to be in the same array, as required by f.ex. vertex arrays.

Function Documentation

void UGLICleanReallocArray (UGLIReallocArray * *reallocarray*)

Cleans the specified ReallocArray (size set to 0).

Parameters:

Returns:

reallocarray	ReallocArray to clean.

void UGLIDeleteReallocArray (UGLIReallocArray * *reallocarray*)

Deletes the specified ReallocArray.

Parameters:

reallocarray	ReallocArray to delete.

void UGLIInitReallocArray (UGLIReallocArray * *reallocarray*, size_t *elementsize*)

Initializes the specified ReallocArray.

Parameters:

reallocarray	Array to initialize.
elementsize	Size of a single element (in bytes)

UGLIReallocArray* UGLINewReallocArray (size_t *elementsize*)

Constructor that creates a new ReallocArray.

Parameters:

elementsize	Size of a single element (in bytes)

New array.

size_t UGLIReallocArrayAppendElement (UGLIReallocArray * *reallocarray*, const void *const *element*)

Appends the specified element to ReallocArray.

Parameters:

Returns:

reallocarray	ReallocArray to append to.
element	Element to append.

New array size (on fail, 0).

size_t UGLIReallocArrayAppendPartialElement (UGLIReallocArray * *reallocarray*, const void *const *element*, size_t *bytes*)

Appends the specified partial element to ReallocArray.

Appends the specified partial element to ReallocArray. All extra bytes of the element will be set to 0.

Parameters:

Returns:

reallocarray	ReallocArray to append to.
element	Element to append.
bytes	Number of bytes to set.

New array size (on fail, 0).

int UGLIReallocArrayFinalize (UGLIReallocArray * *reallocarray*)

Finalizes ReallocArray size.

Parameters:

Returns:

reallocarray	Array finalize.

0 on success.

void* UGLIReallocArrayGetElement (const UGLIReallocArray * *reallocarray*, size_t *item*)

Gets the specified ReallocArray element.

Parameters:

Returns:

reallocarray	ReallocArray to get the element from.
item	Which item to get.

Item.

size_t UGLIReallocArrayHasSpace (UGLIReallocArray * *reallocarray*, size_t *elementsrequired*)

Checks if there's enough space in the specified ReallocArray.

Parameters:

Returns:

reallocarray	Array to check.
elementsrequired	How many more elements need to fit.

Amount of free space (in amount of elements).

void UGLIReallocArrayRemoveElementById (UGLIReallocArray * *reallocarray*, size_t *elementid*)

Removes specified element (by id) from the ReallocArray, and moves further elements back by one step.

Parameters:

Returns:

reallocarray	ReallocArray to remove the element from.
elementid	Element id to remove.

void UGLIReallocArrayRemoveElementsById (UGLIReallocArray * *reallocarray*, size_t *elementid*, size_t *amount*)

Removes number of elements (by id) from the ReallocArray, and moves further elements back by one step.

Parameters:

reallocarray	ReallocArray to remove the elements from.
elementid	Element id of the first element to remove.
amount	Amount of elements to remove.

void UGLIReallocArraySetArray (UGLIReallocArray * *reallocarray*, void * *array*, size_t *elements*)

Replaces the current ReallocArray array with the specified one. The old array will be freed.

Parameters:

reallocarray	ReallocArray to set array for.
array	New array.
elements	New array size in number of elements.

void UGLIReallocArraySetElement (UGLIReallocArray * *reallocarray*, size_t *item*, const void *const *element*)

Sets the specified ReallocArray element.

Sets the specified ReallocArray element. Array will be expanded until specified slot, if needed.

Parameters:

reallocarray	ReallocArray to get the element from.
item	Which item to set.
element	Element to set.

size_t UGLIReallocArraySetSize (UGLIReallocArray * *reallocarray*, size_t *size*)

Sets ReallocArray size to at least specified amount.

Parameters:

reallocarray	Array to set.
size	New size.

Amount of free space (in amount of elements).

---

savepng.h File Reference

Save specified pixel data as a png image.

```
#include "flag_p.h"
#include "header_begin.h"
```

```
#include "header_end.h"
```

Enumerations

enum **UGLIWritePngFlag** { **UGLI_PNG_FLIP_Y** = UGLI_FLAG(0) }

Functions

void **UGLISavePng** (const char *filename, const void *pixeldata, unsigned int width, unsigned int height, unsigned int linestride, unsigned int bitdepth, **UGLIWritePngFlag** flags)

Detailed Description

Save specified pixel data as a png image.

Enumeration Type Documentation

enum UGLIWritePngFlag

PNG saving related flags.

Enumerator:

UGLI_PNG_FLIP_Y Flip image in Y before saving.

Function Documentation

void UGLISavePng (const char * *filename*, const void * *pixeldata*, unsigned int *width*, unsigned int *height*, unsigned int *linestride*, unsigned int *bitdepth*, UGLIWritePngFlag *flags*)

Writes pixels from the specified location to the specified file as png image.

Parameters:

Returns:

filename	File to save to.
pixeldata	Pixel data for the image.
width	Width of the image.
height	Height of the image.
linestride	Amount of bytes between the start of 2 successive lines.
bitdepth	Bits per pixel.
flags	Flags.

---

stringlist.h File Reference

String list.

```
#include <stdlib.h>
#include <stdio.h>
#include "reallocarray.h"
#include "header_begin.h"
#include "header_end.h"
```

Macros

#define **STRINGLIST_NO_EMPTY_LINES** 0x01
#define **STRINGLIST_NO_COMMENT_LINES** 0x02
#define **STRINGLIST_TRIM_SPACES** 0x04

Typedefs

typedef **UGLIReallocArray UGLIStringList**

Functions

UGLIStringList * **UGLINewStringList** (void)
UGLIStringList * **UGLINewStringListFromFP** (FILE *file, unsigned int options)
UGLIStringList * **UGLINewStringListFromFile** (const char *filename, unsigned int options)
void **UGLIInitStringList** (**UGLIStringList** *stringlist)
int **UGLIInitStringListFromFP** (**UGLIStringList** *stringlist, FILE *file, unsigned int options)
int **UGLIInitStringListFromFile** (**UGLIStringList** *stringlist, const char *filename, unsigned int options)
void **UGLIDeleteStringList** (**UGLIStringList** *stringlist)
void **UGLICleanStringList** (**UGLIStringList** *stringlist)
int **UGLIStringListAppendString** (**UGLIStringList** *stringlist, const char *string)
int **UGLIStringListHasString** (**UGLIStringList** *stringlist, const char *string)
char * **UGLIStringListGetString** (**UGLIStringList** *stringlist, unsigned int id)
int **UGLIStringListSaveToFile** (const **UGLIStringList** *stringlist, const char *filename)

Detailed Description

String list.

Component that can contain several strings, all of which can be loaded from a file. It's also possible to manually append strings. User can check if StringList contains specified string, good for white / blacklisting etc.

Macro Definition Documentation

#define STRINGLIST_NO_COMMENT_LINES 0x02

Remove comment lines when loading from a file.

#define STRINGLIST_NO_EMPTY_LINES 0x01

Remove empty lines when loading from file.

#define STRINGLIST_TRIM_SPACES 0x04

Remove extra whitespace from the beginning and the end of line.

Typedef Documentation

typedef UGLIReallocArray UGLIStringList

StringList object.

Function Documentation

void UGLICleanStringList (UGLIStringList * *stringlist*)

Cleans the specified StringList.

Parameters:

stringlist	StringList to clean.

void UGLIDeleteStringList (UGLIStringList * *stringlist*)

Destructor that deletes the specified StringList.

Parameters:

stringlist	StringList to delete.

void UGLIInitStringList (UGLIStringList * *stringlist*)

Initializes specified StringList.

Parameters:

stringlist	StringList to initialize.

int UGLIInitStringListFromFile (UGLIStringList * *stringlist*, const char * *filename*, unsigned int *options*)

Initializes new StringList from a specified.

Parameters:

stringlist	StringList to initialize.

filename	File to load lines from.
options	Options for loading data.

0 on success.

int UGLIInitStringListFromFP (UGLIStringList * *stringlist*, FILE * *file*, unsigned int *options*)

Initializes new StringList from a specified file pointer.

Parameters:

Returns:

stringlist	StringList to initialize.
file	File pointer to load lines from.
options	Options for loading data.

0 on success.

UGLIStringList* UGLINewStringList (void)

Constructor that creates a new empty StringList.

Returns:

new StringList (or NULL if failed)

UGLIStringList* UGLINewStringListFromFile (const char * *filename*, unsigned int *options*)

Constructor that creates a new StringList from a specified file.

Parameters:

Returns:

filename	File to load lines from.
options	Options for loading data.

new StringList (or NULL if failed)

UGLIStringList* UGLINewStringListFromFP (FILE * *file*, unsigned int *options*)

Constructor that creates a new StringList from a specified file pointer.

Parameters:

Returns:

file	File pointer to load lines from.
options	Options for loading data.

new StringList (or NULL if failed)

int UGLIStringListAppendString (UGLIStringList * *stringlist*, const char * *string*)

Appends the specified string to StringList.

Parameters:

Returns:

stringlist	StringList to append string to.
string	String to append.

???

char* UGLIStringListGetString (UGLIStringList * *stringlist*, unsigned int *id*)

Gets the specified string from the specified StringList.

Parameters:

Returns:

stringlist	StringList to get string from.
id	Which string to get.

Requested string.

int UGLIStringListHasString (UGLIStringList * *stringlist*, const char * *string*)

Checks whether the specified StringList has the specified string.

Parameters:

Returns:

stringlist	StringList to check.
string	String to find.

index of first match if found, otherwise -1

int UGLIStringListSaveToFile (const UGLIStringList * *stringlist*, const char * *filename*)

Saves the specified StringList to file.

Parameters:

Returns:

stringlist	StringList to save.
filename	File to save to.

0 on success

---

texture.h File Reference

Texture related functionality.

```
#include "window.h"
#include "image.h"
#include "header_begin.h"
#include "header_end.h"
```

Macros

#define **UGLI_NO_TEXTURE** NULL

Typedefs

typedef struct UGLITextureData * **UGLITexture**

Enumerations

enum **UGLIPixelFormat** { **UGLI_RGB**, **UGLI_RGBA** }

enum **UGLITextureFlag** { **UGLI_F_TEXTURE_NONE** = 0x0000,
UGLI_F_TEXTURE_KEEP_PIXELS = 0x0001 }
enum **UGLIRenderToTextureFlag** { **UGLI_F_RTT_NONE** = 0x0000,
UGLI_F_RTT_USE_DEPTH = 0x0001 }

Functions

UGLITexture UGLINewTextureFromFile (const char *const filename,
UGLITextureFlag flags)
Loads specified file as a texture. Uses already loaded previous instance, if possible.

UGLITexture UGLINewTextureFromImage (**UGLIImage** image,
UGLITextureFlag flags)
Creates a texture of the specified Image.

UGLITexture UGLINewTextureEmpty (int width, int height, **UGLIPixelFormat** format)
Creates a new, empty texture.

void **UGLIDeleteTexture** (**UGLITexture** texture)
Destructor for texture object.

void **UGLITextureSetSize** (**UGLITexture** texture, int width, int height,
UGLIPixelFormat format)
Sets texture size & pixel format.

void **UGLITextureBind** (const **UGLITexture** texture)
Binds specified texture.

unsigned char **UGLITextureGetAlpha** (const **UGLITexture** texture, unsigned int x, unsigned int y)
Returns alpha value of the specified pixel in texture, if possible.

UGLITexture UGLIRenderToTextureBegin (**UGLITexture** texture,
UGLIRenderToTextureFlag flags, unsigned int xpos, unsigned int ypos, unsigned int width, unsigned int height)
Starts rendering to the specified texture.

void **UGLIRenderToTextureEnd** (**UGLITexture** texture)
Ends rendering to the specified texture.

Detailed Description

Texture related functionality.

Macro Definition Documentation

#define UGLI_NO_TEXTURE NULL

Specify for UGLITextureBind to unbind texture

Typedef Documentation

typedef struct UGLITextureData* UGLITexture

UGLITexture object.

Enumeration Type Documentation

enum UGLIPixelFormat

Specify pixel format.

Enumerator:

UGLI_RGB RGB pixel format. 3 bytes per pixel.
UGLI_RGBA RGBA pixel format. 4 bytes per pixel.

enum UGLIRenderToTextureFlag

Render to texture option flags.

Enumerator:

UGLI_F_RTT_NONE No flags.
UGLI_F_RTT_USE_DEPTH Depth buffer required for correct drawing.

enum UGLITextureFlag

Flags to determine texture handling.

Enumerator:

UGLI_F_TEXTURE_NONE No flags.
UGLI_F_TEXTURE_KEEP_PIXELS Keep pixels in ram after creating the texture.

Function Documentation

void UGLIDeleteTexture (UGLITexture *texture*)

Destructor for texture object.

Parameters:

Returns:

texture	Texture object to delete.

UGLITexture UGLINewTextureEmpty (int *width*, int *height*, UGLIPixelFormat *format*)

Creates a new, empty texture.

Parameters:

width	Width of the texture.
height	Height of the texture.
format	Pixel format.

UGLITexture UGLINewTextureFromFile (const char *const *filename*, UGLITextureFlag *flags*)

Loads specified file as a texture. Uses already loaded previous instance, if possible.

Parameters:

filename	File to load texture data from.
flags	Flags.

UGLITexture UGLINewTextureFromImage (UGLIImage *image*, UGLITextureFlag *flags*)

Creates a texture of the specified Image.

Parameters:

image	Image to create texture from.
flags	Flags.

UGLITexture UGLIRenderToTextureBegin (UGLITexture *texture*, UGLIRenderToTextureFlag *flags*, unsigned int *xpos*, unsigned int *ypos*, unsigned int *width*, unsigned int *height*)

Starts rendering to the specified texture.

Starts rendering to the specified texture. While rendering, the texture cannot be used.

Parameters:

texture	Texture to render to.
flags	Rendering options.
xpos	X position of viewport.
ypos	Y position of viewport.
width	Width of viewport.
height	Height of viewport.

UGLITexture user wishes to render to (or NULL on fail)

void UGLIRenderToTextureEnd (UGLITexture *texture*)

Ends rendering to the specified texture.

Parameters:

Returns:

texture	Texture to end rendering to.

void UGLITextureBind (const UGLITexture *texture*)

Binds specified texture.

Parameters:

texture	Texture object to bind (or NULL for none).

unsigned char UGLITextureGetAlpha (const UGLITexture *texture*, unsigned int *x*, unsigned int *y*)

Returns alpha value of the specified pixel in texture, if possible.

Parameters:

texture	Texture object to get the alpha value from.
x	X coordinate of the pixel.
y	Y coordinate of the pixel.

Alpha value (or 0 if not possible)

void UGLITextureSetSize (UGLITexture *texture*, int *width*, int *height*, UGLIPixelFormat *format*)

Sets texture size & pixel format.

Parameters:

Returns:

texture	Texture to set size for.
width	Width for the texture.
height	Height for the texture.
format	Pixel format.

---

threads.h File Reference

Thread management.

```
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct UGLIThreadData * **UGLIThread**

Functions

UGLIThread UGLINewThread (int(*function)(void *), void *data)
void **UGLIThreadWaitThread** (**UGLIThread** thread, int *status)

Detailed Description

Thread management.
Functions for handling several threads.

Typedef Documentation

typedef struct UGLIThreadData* UGLIThread

Thread object.

Function Documentation

UGLIThread UGLINewThread (int(*)(void *) *function*, void * *data*)

Constructor. Creates new thread that will run the specified function.

Parameters:

function	Function the thread should run.
data	Data passed to the function.

New GenThread object.

void UGLIThreadWaitThread (UGLIThread *thread*, int * *status*)

Destructor. Waits for the specified thread to finish.

Parameters:

Returns:

thread	GenThread to wait for.
status	Return value from the thread function.

---

time.h File Reference

Time-related functionality.

```
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef unsigned int **UGLITicks**

Functions

UGLITicks UGLIGetTicks (void)

Returns the amount of ticks since program start.

void **UGLIDelay** (**UGLITicks** msecs)

Delays thread execution for specified amount of ticks.

UGLITicks UGLIGetDelta (**UGLITicks** *last, **UGLITicks** newticks)

Gets delta between old and new time. Updates old to new.

Detailed Description

Time-related functionality.

Typedef Documentation

typedef unsigned int UGLITicks

Ticks (milliseconds) type.

Function Documentation

void UGLIDelay (UGLITicks *msecs*)

Delays thread execution for specified amount of ticks.

Parameters:

msecs	Amount of milliseconds to wait.

UGLITicks UGLIGetDelta (UGLITicks * *last*, UGLITicks *newticks*)

Gets delta between old and new time. Updates old to new.

Parameters:

last	Pointer to last ticks value (will be updated to newticks).
newticks	New ticks value.

Delta ticks.

UGLITicks UGLIGetTicks (void)

Returns the amount of ticks since program start.

Returns:

Amount of ticks since program start.

---

uielements/glframe.h File Reference

Simple frame object.

```
#include "../texture.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Typedefs

typedef struct UGLIUIFrameData * **UGLIUIFrame**

Functions

UGLIUIFrame UGLIUINewFrameFromTexturePart (**UGLITexture** texture, int xpmin, int xpmax, int ypmin, int ypmax, int xminmargin, int xmaxmargin, int yminmargin, int ymaxmargin)
UGLIUIFrame UGLIUINewFrame (const char *const filename, int xminmargin, int xmaxmargin, int yminmargin, int ymaxmargin)
void **UGLIUIDeleteFrame** (**UGLIUIFrame** frame)
void **UGLIUIFrameSetViewport** (**UGLIUIFrame** frame, unsigned int xpos, unsigned int ypos, unsigned int width, unsigned int height)
void **UGLIUIFrameSetColor4FV** (**UGLIUIFrame** frame, const float *color)
void **UGLIUIFrameSetOpacity** (**UGLIUIFrame** frame, float opacity)
void **UGLIUIFramePrepareDraw** (void)
void **UGLIUIFrameDraw** (const **UGLIUIFrame** frame)

Detailed Description

Simple frame object.

Graphical frame object, that can be resized to any size.

Typedef Documentation

typedef struct UGLIUIFrameData* UGLIUIFrame

UGLIUIFrame object.

Function Documentation

void UGLIUIDeleteFrame (UGLIUIFrame *frame*)

Destructor that deletes the specified UGLIUIFrame.

Parameters:

Returns:

frame	UGLIUIFrame to delete.

void UGLIUIFrameDraw (const UGLIUIFrame *frame*)

Draws specified UGLIUIFrame.

Parameters:

frame	UGLIUIFrame to draw.

void UGLIUIFramePrepareDraw (void)

Prepare drawing any amount of UGLIUIFrames to current window.

void UGLIUIFrameSetColor4FV (UGLIUIFrame *frame*, const float * *color*)

Set UGLIUIFrame color.

Parameters:

frame	UGLIUIFrame to set color for.
color	pointer to float[4] that defines new RGBA color. If NULL, using default color (full white) instead.

void UGLIUIFrameSetOpacity (UGLIUIFrame *frame*, float *opacity*)

Set UGLIUIFrame opacity.

Parameters:

frame	UGLIUIFrame to set opacity for.
opacity	New opacity (0.0-1.0).

void UGLIUIFrameSetViewport (UGLIUIFrame *frame*, unsigned int *xpos*, unsigned int *ypos*, unsigned int *width*, unsigned int *height*)

Set UGLIUIFrame viewport.

Parameters:

frame	UGLIUIFrame to set.
xpos	X position of the viewport.
ypos	Y position of the viewport.

width	Width of the viewport.
height	Height of the viewport.

UGLIUIFrame UGLIUINewFrame (const char *const *filename*, int *xminmargin*, int *xmaxmargin*, int *yminmargin*, int *ymaxmargin*)

Constructor that creates a new UGLIUIFrame.

Parameters:

filename	File to load texture from.
xminmargin	X min margin of the frame.
xmaxmargin	X max margin of the frame.
yminmargin	Y min margin of the frame.
ymaxmargin	Y max margin of the frame.

new UGLIUIFrame object (if successful, otherwise NULL)

UGLIUIFrame UGLIUINewFrameFromTexturePart (UGLITexture *texture*, int *xpmin*, int *xpmax*, int *ypmin*, int *ypmax*, int *xminmargin*, int *xmaxmargin*, int *yminmargin*, int *ymaxmargin*)

Constructor that creates a new UGLIUIFrame from a specified part of loaded texture.

Parameters:

Returns:

texture	Texture to use.
xpmin	X min position of the bounding box.
xpmax	X max position of the bounding box.
ypmin	Y min position of the bounding box.
ypmax	Y max position of the bounding box.
xminmargin	X min margin of the frame.
xmaxmargin	X max margin of the frame.
yminmargin	Y min margin of the frame.
ymaxmargin	Y max margin of the frame.

new UGLIUIFrame object (if successful, otherwise NULL)

---

uielements/glstring.h File Reference

Simple string object.

```
#include <stdarg.h>
#include "../glfont.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Typedefs

typedef struct UGLIUIStringData * **UGLIUIString**

Enumerations

enum **UGLIUITextDirection** { **UGLIUI_TEXT_DIRECTION_HORIZONTAL**, **UGLIUI_TEXT_DIRECTION_VERTICAL** }

Functions

UGLIUIString UGLIUINewString (const **UGLIFont** font, **UGLIUITextDirection** direction, unsigned int maxlength, const float *const color)
UGLIUIString NewGLStringWithText (const **UGLIFont** font, **UGLIUITextDirection** direction, const char *const str, const float *const color)
void **UGLIUIDeleteString** (**UGLIUIString** str)
void **UGLIUIStringSetText** (**UGLIUIString** str, const char *const newtext)
void **UGLIUIStringSetTextFormatted** (**UGLIUIString** str, const char *format,...)
void **UGLIUIStringSetTextFormattedVA** (**UGLIUIString** str, const char *format, va_list args)
void **UGLIUIStringSetInt** (**UGLIUIString** str, int number)
void **UGLIUIStringSetUInt** (**UGLIUIString** str, unsigned int number)
void **UGLIUIStringSetFloat** (**UGLIUIString** str, float number)
void **UGLIUIStringSetCharacter** (**UGLIUIString** str, unsigned int index, char ch)
void **UGLIUIStringSetColor** (**UGLIUIString** str, const float *const color)
void **UGLIUIStringSetMaxDisplayLen** (**UGLIUIString** str, unsigned int maxdisplaylength)
void **CleanGLStringText** (**UGLIUIString** str)
float **UGLIUIStringGetWidth** (**UGLIUIString** str)
float **UGLIUIStringGetHeight** (**UGLIUIString** str)
void **UGLIUIStringPrepareDraw** (void)
void **UGLIUIStringDraw** (const **UGLIUIString** str)

Detailed Description

Simple string object.

Typedef Documentation

typedef struct UGLIUIStringData* UGLIUIString

UGLIUIString object.

Enumeration Type Documentation

enum UGLIUITextDirection

Text direction.

Enumerator:

UGLIUI_TEXT_DIRECTION_HORIZONTAL Text should be drawn horizontally.
UGLIUI_TEXT_DIRECTION_VERTICAL Text should be drawn vertically.

Function Documentation

void CleanGLStringText (UGLIUIString *str*)

Sets specified constant string length to 0.

Parameters:

Returns:

str	String to set

UGLIUIString NewGLStringWithText (const UGLIFont *font*, UGLIUITextDirection *direction*, const char *const *str*, const float *const *color*)

Convenience function. Creates new constant string ID, and assigns given string to it.

Parameters:

font	Font to use
direction	Which direction the string should be drawn in (h/v).
str	Set constant string text to this
color	Color to use. NULL uses full white.

new string ID

void UGLIUIDeleteString (UGLIUIString *str*)

Deletes given string.

Parameters:

Returns:

str	string to delete.

UGLIUIString UGLIUINewString (const UGLIFont *font*, UGLIUITextDirection *direction*, unsigned int *maxlength*, const float *const *color*)

Creates new constant string ID.

Parameters:

font	Font to use.
direction	Which direction the string should be drawn in (h/v).
maxlength	max allowed string length.
color	Color to use. NULL uses full white.

new string

void UGLIUIStringDraw (const UGLIUIString *str*)

Draws specified constant string at origo.

Parameters:

Returns:

str	String to draw.

float UGLIUIStringGetHeight (UGLIUIString *str*)

Gets string height.

Parameters:

str	String to use

size of requested dimension

float UGLIUIStringGetWidth (UGLIUIString *str*)

Gets string width.

Parameters:

Returns:

str	String to use

size of requested dimension

void UGLIUIStringPrepareDraw (void)

Do required preparations for drawing text.

void UGLIUIStringSetCharacter (UGLIUIString *str*, unsigned int *index*, char *ch*)

Sets string text to a single character.

Parameters:

Returns:

str	String to set.
index	Which character to change
ch	character to which the string should be set to

void UGLIUIStringSetColor (UGLIUIString *str*, const float *const *color*)

Sets constant string color.

Parameters:

str	String to set.
color	Color to set (pointer to float[4]). NULL for white.

void UGLIUIStringSetFloat (UGLIUIString *str*, float *number*)

Sets string text to specified float.

Parameters:

str	String to set.
number	float to which the string should be set to.

void UGLIUIStringSetInt (UGLIUIString *str*, int *number*)

Sets string text to specified integer.

Parameters:

str	String to set.
number	integer to which the string should be set to.

void UGLIUIStringSetMaxDisplayLen (UGLIUIString *str*, unsigned int *maxdisplaylength*)

Sets constant string maximum display length.

Parameters:

str	String to set.
maxdisplaylength	New maximum display length.

void UGLIUIStringSetText (UGLIUIString *str*, const char *const *newtext*)

Sets string text.

Parameters:

str	String to set.
newtext	new string for constant string.

void UGLIUIStringSetTextFormatted (UGLIUIString *str*, const char * *format*, ...)

Sets string formatted text.

Parameters:

str	String to set.
format	Format string.
...	Other args.

void UGLIUIStringSetTextFormattedVA (UGLIUIString *str*, const char * *format*, va_list *args*)

Sets string formatted text with varargs.

Parameters:

str	String to set.
format	Format string.
args	varargs.

void UGLIUIStringSetUInt (UGLIUIString *str*, unsigned int *number*)

Sets string text to specified unsigned integer.

Parameters:

str	String to set.
number	unsigned integer to which the string should be set to.

---

uielements/log.h File Reference

simple text log component.

```
#include <stdarg.h>
#include "../window.h"
#include "../glfont.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Typedefs

typedef struct UGLIUILogData * **UGLIUILog**

Functions

UGLIUILog UGLIUINewLog (**UGLIFont** font, unsigned int lines, unsigned int maxlinelen)
void **UGLIUIDeleteLog** (**UGLIUILog** logitem)
void **UGLIUILogSetViewport** (**UGLIUILog** logitem, unsigned int xpos, unsigned int ypos, unsigned int width, unsigned int height)
void **UGLIUILogDraw** (**UGLIUILog** logitem)
void **UGLIUILogAddText** (**UGLIUILog** logitem, const char *text)
void **UGLIUILogAddTextFormattedVA** (**UGLIUILog** logitem, const char *string, va_list args)
void **UGLIUILogAddTextFormatted** (**UGLIUILog** logitem, const char *format,...)

Detailed Description

simple text log component.

User can log lines of text to the component, and it will draw specified amount of latest lines.

Component is designed to use with ortho projection, but it isn't enforced in any way, the user is free to use any kind of transformations he wants.

LogLI component is a LayoutItem version, which enforces this.

Typedef Documentation

typedef struct UGLIUILogData* UGLIUILog

UGLIUILog object.

Function Documentation

void UGLIUIDeleteLog (UGLIUILog *logitem*)

Deletes specified UGLIUILog component.

Parameters:

logitem	UGLIUILog to delete.

void UGLIUILogAddText (UGLIUILog *logitem*, const char * *text*)

Logs text to the specified UGLIUILog component.

Parameters:

logitem	UGLIUILog to log to.

text	Text to log.

void UGLIUILogAddTextFormatted (UGLIUILog *logitem*, const char * *format*, ...)

Logs text to the specified GenLog component.

Parameters:

logitem	GenLog to log to.
format	Format string.
...	Other args.

void UGLIUILogAddTextFormattedVA (UGLIUILog *logitem*, const char * *string*, va_list *args*)

Logs text to the specified GenLog component.

Parameters:

logitem	GenLog to log to.
string	Control string.
args	va_list.

void UGLIUILogDraw (UGLIUILog *logitem*)

Draws specified UGLIUILog component.

Parameters:

logitem	UGLIUILog to draw.

void UGLIUILogSetViewport (UGLIUILog *logitem*, unsigned int *xpos*, unsigned int *ypos*, unsigned int *width*, unsigned int *height*)

Sets UGLIUILog viewport.

Parameters:

logitem	UGLIUILog to modify.
xpos	New x position.
ypos	New y position.
width	New width.
height	New height.

UGLIUILog UGLIUINewLog (UGLIFont *font*, unsigned int *lines*, unsigned int *maxlinelen*)

Creates new UGLIUILog component.

Parameters:

font	Font to use.
lines	Maximum number of last lines to store.
maxlinelen	Maximum length (in chars) for each line.

new GenLog (or NULL if not successful).

uielements/logli.h File Reference

LayoutItem wrapper for Log component.

```
#include "../layout/layoutitem.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Functions

UGLILayoutItem UGLIUINewLogLI (**UGLIUILog** log)

Detailed Description

LayoutItem wrapper for Log component.

Function Documentation

UGLILayoutItem UGLIUINewLogLI (UGLIUILog *log*)

Creates Log LayoutItem.

Parameters:

log	GenLog to use.

Returns:

new LayoutItem (or NULL if not successful).

---

uielements/textureli.h File Reference

simple texture UGLILayoutItem component.

```
#include "../texture.h"
#include "../layout/layoutitem.h"
#include "../header_begin.h"
#include "../header_end.h"
```

Functions

UGLILayoutItem UGLIUINewTextureLI (**UGLITexture** texture)

Detailed Description

simple texture UGLILayoutItem component.

Function Documentation

UGLILayoutItem UGLIUINewTextureLI (UGLITexture *texture*)

Creates new UGLIUITexture component with the specified texture.

Parameters:

Returns:

texture	UGLITexture to use for the item.

new UGLIUITexture (if successful, otherwise NULL)

---

vectormath2d.h File Reference

2D vector math routines

```
#include "header_begin.h"
#include "header_end.h"
```

Functions

float **UGLIVec2DLen** (const float *v)
float **UGLIVec2DLenSquare** (const float *v)
void **UGLIVec2DNormalize** (float *v)
float **UGLIDistance2D** (const float *a, const float *b)
float **UGLIDistance2DSquare** (const float *a, const float *b)
float **UGLIDotProduct2D** (const float *a, const float *b)

Detailed Description

2D vector math routines

Function Documentation

float UGLIDistance2D (const float * *a*, const float * *b*)

Returns the distance between 2 specified 2D points.

Parameters:

Returns:

a	Pointer to float[2] that specifies the first point.
b	Pointer to float[2] that specifies the second point.

The distance between the 2 points if successful (otherwise NULL)

float UGLIDistance2DSquare (const float * *a*, const float * *b*)

Returns the square of the distance between 2 specified 2D points.

Parameters:

Returns:

a	Pointer to float[2] that specifies the first point.
b	Pointer to float[2] that specifies the second point.

The square of the distance between the 2 points if successful (otherwise NULL)

float UGLIDotProduct2D (const float * *a*, const float * *b*)

Returns the dot product between the two specified 2D vectors.

Parameters:

Returns:

a	Pointer to the first float[2] vector.
b	Pointer to the second float[2] vector.

The dot product of the two specified vectors.

float UGLIVec2DLen (const float * *v*)

Returns the length of the specified 2D vector.

Parameters:

Returns:

v	Pointer to float[2] that specifies the vector.

Length of the specified vector if successful (otherwise NULL)

float UGLIVec2DLenSquare (const float * *v*)

Returns pow2 of the length of the specified 2D vector.

Parameters:

Returns:

v	Pointer to float[2] that specifies the vector.

pow2 length of the specified vector if successful (otherwise NULL)

void UGLIVec2DNormalize (float * *v*)

Normalizes the specified 2D vector to 1.0f length.

Parameters:

Returns:

v	Pointer to float[2] vector to normalize.

---

vectormath3d.h File Reference

3D vector math routines

```
#include "angle.h"
#include "header_begin.h"
#include "header_end.h"
```

Functions

float **UGLIVec3DLen** (const float *v)
float **UGLIVec3DLenSquare** (const float *v)
float **UGLIDistance3D** (const float *a, const float *b)
float **UGLIDistance3DSquare** (const float *a, const float *b)
void **UGLIVec3DNormalize** (float *v)
float **UGLIVec3DAngleRad** (const float *a, const float *b)
float **UGLIDotProduct3D** (const float *a, const float *b)
void **UGLICrossProduct3D** (float *result, const float *a, const float *b)

Detailed Description

3D vector math routines

Function Documentation

void UGLICrossProduct3D (float * *result*, const float * *a*, const float * *b*)

Returns the cross product between the two specified 3D vectors.

Parameters:

result	Pointer to the float[3] where the result vector should be stored.
a	Pointer to the first float[3] vector.
b	Pointer to the second float[3] vector.

float UGLIDistance3D (const float * *a*, const float * *b*)

Returns the distance between 2 specified 3D points.

Parameters:

a	Pointer to float[3] that specifies the first point.
b	Pointer to float[3] that specifies the second point.

The distance between the 2 points if successful (otherwise NULL)

float UGLIDistance3DSquare (const float * *a*, const float * *b*)

Returns the square of the distance between 2 specified 3D points.

Parameters:

Returns:

a	Pointer to float[3] that specifies the first point.
b	Pointer to float[3] that specifies the second point.

The square of the distance between the 2 points if successful (otherwise NULL)

float UGLIDotProduct3D (const float * *a*, const float * *b*)

Returns the dot product between the two specified 3D vectors.

Parameters:

Returns:

a	Pointer to the first float[3] vector.
b	Pointer to the second float[3] vector.

The dot product of the two specified vectors.

float UGLIVec3DAngleRad (const float * *a*, const float * *b*)

Returns the angle between the two specified direction vectors in radians.

Parameters:

Returns:

a	Pointer to the first float[3] direction vector.
b	Pointer to the second float[3] direction vector.

The angle between the two direction vectors in radians.

Returns the angle between 2 3D vectors in radians.

Parameters:

Returns:

a	Pointer to the first float[3] vector.
b	Pointer to the second float[3] vector.

Angle in radians.

float UGLIVec3DLen (const float * *v*)

Returns the length of the specified 3D vector.

Parameters:

Returns:

v	Pointer to float[3] that specifies the vector.

Length of the specified vector if successful (otherwise NULL)

float UGLIVec3DLenSquare (const float * *v*)

Returns the square of the length of the specified 3D vector.

Parameters:

Returns:

v	Pointer to float[3] that specifies the vector.

The square of the length of the specified vector if successful (otherwise NULL)

void UGLIVec3DNormalize (float * *v*)

Normalizes the specified 3D vector to 1.0f length.

Parameters:

Returns:

v	Pointer to float[3] vector to normalize.

---

vertexarray.h File Reference

Vertex array management.

```
#include <stddef.h>
#include "flag_p.h"
#include "texture.h"
#include "glprimitive.h"
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct
UGLIVertexArrayData * **UGLIVertexArray**

Enumerations

enum **UGLIVertexArrayFlag** { **UGLI_VA_STATIC_VERTICES** = UGLI_FLAG(0), **UGLI_VA_STATIC_LENGTH** = UGLI_FLAG(1), **UGLI_VA_APPEND_INDEXED** = UGLI_FLAG(2), **UGLI_VA_OPTIMIZE_SPEED_BIG_SIZE** = UGLI_FLAG(3), **UGLI_VA_GET_RADIUS** = UGLI_FLAG(4), **UGLI_VA_TEXCOORD_2F** = UGLI_FLAG(5), **UGLI_VA_COLOR_4F** = UGLI_FLAG(6), **UGLI_VA_NORMAL_3F** = UGLI_FLAG(7) }
enum **UGLIVASetArraysFlag** { **UGLI_VA_SET_ARRAY_FINISH** = UGLI_FLAG(0), **UGLI_VA_SET_ARRAY_NO_COPY_VA** = UGLI_FLAG(1), **UGLI_VA_SET_ARRAY_NO_COPY_IA** = UGLI_FLAG(2) }
enum **UGLIVertexArrayType** { **UGLI_VA_ARRAY_VERTEX** = 0, **UGLI_VA_ARRAY_INDEX** = 1 }

Functions

UGLIVertexArray UGLINewVertexArray (**UGLIPrimitive** primitive, **UGLIVertexArrayFlag** flags)
void **UGLIDeleteVertexArray** (**UGLIVertexArray** vertexarray)
void **UGLIVertexArrayFinish** (**UGLIVertexArray** vertexarray)
int **UGLIVertexArrayCreateIndexes** (**UGLIVertexArray** vertexarray)
void **UGLIVertexArrayDraw** (const **UGLIVertexArray** vertexarray)
void **UGLIVertexArrayDrawRange** (const **UGLIVertexArray** vertexarray, unsigned int first, unsigned int count)
void **UGLIVertexArrayDrawRanges** (const **UGLIVertexArray** vertexarray, unsigned int ranges, const unsigned int *first, const int *count)

void **UGLIVertexArrayDrawRangesStride** (const **UGLIVertexArray** m, unsigned int vperrange, unsigned int ranges, unsigned int first, unsigned int stride)
void **UGLIVertexArraySetTexture** (**UGLIVertexArray** vertexarray, **UGLITexture** texture)
void **UGLIVertexArraySetArrays** (**UGLIVertexArray** vertexarray, const float *varray, unsigned int velements, const unsigned int *iarray, unsigned int ielements)
void **UGLIVertexArrayUpdateVertices** (**UGLIVertexArray** vertexarray, unsigned int first, unsigned int count, **UGLIVertexArrayType** type)
unsigned int **UGLIVertexArrayGetVertexStride** (const **UGLIVertexArray** vertexarray)
int **UGLIVertexArrayAppendVertex** (**UGLIVertexArray** vertexarray, const float *vertex)
unsigned int **UGLIVertexArrayGetNextIndex** (const **UGLIVertexArray** vertexarray)
unsigned int **UGLIVertexArrayGetNumVertices** (const **UGLIVertexArray** vertexarray)
unsigned int **UGLIVertexArrayGetElementSize** (const **UGLIVertexArray** vertexarray)
float **UGLIVertexArrayGetVertex** (const **UGLIVertexArray** vertexarray, size_t vertex, size_t element)
void **UGLIVertexArrayDrawNormals** (const **UGLIVertexArray** vertexarray)

Detailed Description

Vertex array management.

Functions for creating and drawing vertices in most efficient way available.

Typedef Documentation

typedef struct UGLIVertexArrayData* UGLIVertexArray

Vertex array object.

Enumeration Type Documentation

enum UGLIVASetArraysFlag

Enumerator:

UGLI_VA_SET_ARRAY_FINISH Finish UGLIVertexArray after setting flags.
UGLI_VA_SET_ARRAY_NO_COPY_VA Don't copy vertex array, transfer ownership to the UGLIVertexArray object.
UGLI_VA_SET_ARRAY_NO_COPY_IA Don't copy index array, transfer ownership to the UGLIVertexArray object.

enum UGLIVertexArrayFlag

Flags that can be given to UGLIVertexArray constructor.

Enumerator:

UGLI_VA_STATIC_VERTICES Static vertices, will be changed rarely, if ever.
UGLI_VA_STATIC_LENGTH Amount of vertices is static.
UGLI_VA_APPEND_INDEXED MeshBuilder will add new vertices indexed (checking if vertex already exists).
UGLI_VA_OPTIMIZE_SPEED_BIG_SIZE Optimize mesh drawing by potentially using more memory.
UGLI_VA_GET_RADIUS Calculate mesh radius when finalized.
UGLI_VA_TEXCOORD_2F Vertex array should have per-vertex texture coordinates.
UGLI_VA_COLOR_4F Vertex array should have per-vertex colors.
UGLI_VA_NORMAL_3F Vertex array should have per-vertex normals.

enum UGLIVertexArrayType

Enables user to define array type.

Enumerator:

UGLI_VA_ARRAY_VERTEX Vertex array.
UGLI_VA_ARRAY_INDEX Index array.

Function Documentation

void UGLIDeleteVertexArray (UGLIVertexArray *vertexarray*)

Destructor. Deletes specified vertex array.

Parameters:

vertexarray	Vertex array to delete.

UGLIVertexArray UGLINewVertexArray (UGLIPrimitive *primitive*, UGLIVertexArrayFlag *flags*)

Constructor. Creates new vertex array with specified primitive and options.

Parameters:

primitive	The type vertex array should be drawn as.
flags	Flags that define various features.

New vertex array object (or NULL on fail)

int UGLIVertexArrayAppendVertex (UGLIVertexArray *vertexarray*, const float * *vertex*)

Appends vertex to the specified UGLIVertexArray.

Parameters:

Returns:

vertexarray	Array to append to.
vertex	Vertex to append.

0 on success.

int UGLIVertexArrayCreateIndexes (UGLIVertexArray *vertexarray*)

Create indexes (based on vertex array contents) for the specified vertex array. May reduce the amount of vertices.

Parameters:

Returns:

vertexarray	vertex array to create indexes for.

void UGLIVertexArrayDraw (const UGLIVertexArray *vertexarray*)

Draws specified vertex array.

Parameters:

vertexarray	Vertex array to draw.

void UGLIVertexArrayDrawNormals (const UGLIVertexArray *vertexarray*)

Extremely slow debug function, that draws all vertex normals for the specified UGLIVertexArray. NOTE: Will NOT work if VBO support is enabled!

Parameters:

vertexarray	Vertex array to draw the normals for.

void UGLIVertexArrayDrawRange (const UGLIVertexArray *vertexarray*, unsigned int *first*, unsigned int *count*)

Draws single range of vertices from the specified vertex array.

Parameters:

vertexarray	Vertex array to draw.
first	First vertex to draw.
count	Total amount of vertices to draw.

void UGLIVertexArrayDrawRanges (const UGLIVertexArray *vertexarray*, unsigned int *ranges*, const unsigned int * *first*, const int * *count*)

Draws multiple ranges of vertices from the specified vertex array.

Parameters:

vertexarray	Vertex array to draw.
ranges	How many ranges to draw.
first	Pointer to an array, which contains the list of first vertices.

count	Pointer to an array, which contains the list of counts.

void UGLIVertexArrayFinish (UGLIVertexArray *vertexarray*)

Finishes UGLIVertexArray. After this, it is ready to be drawn. MUST be executed from the main thread!

Parameters:

vertexarray	vertex array to finish.

unsigned int UGLIVertexArrayGetElementSize (const UGLIVertexArray *vertexarray*)

Gets vertex element size (in bytes) of the specified UGLIVertexArray.

Parameters:

vertexarray	Array to get the size from.

Vertex size (in bytes).

unsigned int UGLIVertexArrayGetNextIndex (const UGLIVertexArray *vertexarray*)

Gets the next free index in the specified UGLIVertexArray.

Parameters:

Returns:

vertexarray	Array to get index from.

Next free index.

unsigned int UGLIVertexArrayGetNumVertices (const UGLIVertexArray *vertexarray*)

Gets the amount of vertices in the specified UGLIVertexArray.

Parameters:

Returns:

vertexarray	Array to get amount from.

Amount of vertices.

float UGLIVertexArrayGetVertex (const UGLIVertexArray *vertexarray*, size_t *vertex*, size_t *element*)

Gets the specified element of the specified vertex from the specified UGLIVertexArray.

Parameters:

Returns:

vertexarray	Array to get the element from.
vertex	Vertex to get the element from.
element	Element id to get.

Element.

unsigned int UGLIVertexArrayGetVertexStride (const UGLIVertexArray *vertexarray*)

Gets vertex stride of the specified UGLIVertexArray.

Parameters:

Returns:

vertexarray	Array to get stride from.

void UGLIVertexArraySetArrays (UGLIVertexArray *vertexarray*, const float * *varray*, unsigned int *velements*, const unsigned int * *iarray*, unsigned int *ielements*)

Sets UGLIVertexArray arrays.

Parameters:

vertexarray	Vertex array to set Arrays for.
varray	Vertex array, that contains vertex, texcoord, color and normal coordinates (combination of what is used).
velements	Amount of vertices in previous array.
iarray	Index array, that contain indexes for earlier vertex array. NULL for none.
ielements	Amount of elements in index array.

void UGLIVertexArraySetTexture (UGLIVertexArray *vertexarray*, UGLITexture *texture*)

Sets vertex array texture.

Parameters:

vertexarray	Vertex array to set texture for.
texture	New texture (or NULL for none).

void UGLIVertexArrayUpdateVertices (UGLIVertexArray *vertexarray*, unsigned int *first*, unsigned int *count*, UGLIVertexArrayType *type*)

Updates UGLIVertexArray array.

Parameters:

vertexarray	Vertex array to update Array for.
first	index of the first element to update.
count	Amount of elements to update.
type	Which array to update (vertex / index).

---

vertexsizes.h File Reference

Vertex sizes.

```
#include "header_begin.h"
#include "header_end.h"
```

Macros

#define **UGLI_FLOATS_T** 2
#define **UGLI_FLOATS_C** 4
#define **UGLI_FLOATS_N** 3

#define **UGLI_FLOATS_V** 4

Detailed Description

Vertex sizes.
Defines sizes for various vertex data.

Macro Definition Documentation

#define UGLI_FLOATS_C 4

Amount of vertices per color.

#define UGLI_FLOATS_N 3

Amount of vertices per normal.

#define UGLI_FLOATS_T 2

Amount of vertices per texcoord.

#define UGLI_FLOATS_V 4

Amount of vertices per vertex.

---

window.h File Reference

Window management.

```
#include "physics.h"
#include "maincontext.h"
#include "events.h"
#include "header_begin.h"
#include "header_end.h"
```

Typedefs

typedef struct UGLIWindowData * **UGLIWindow**

Enumerations

enum **UGLIWindowTag** { **UGLI_T_WINDOW_DONE**,
UGLI_T_WINDOW_XPOS, **UGLI_T_WINDOW_YPOS**,
UGLI_T_WINDOW_WIDTH, **UGLI_T_WINDOW_HEIGHT**,
UGLI_T_WINDOW_TITLE, **UGLI_T_WINDOW_RESHAPE**,
UGLI_T_WINDOW_DRAWCONTENTS,

UGLI_T_WINDOW_HANDLEEVENT,
UGLI_T_WINDOW_CLEARCOLOR }
Tags that can be given to UGLIWindowOpen to give window specific properties. enum **UGLIWinClearFlag** {
UGLI_F_WINDOW_CLEAR_COLOR = 0x01,
UGLI_F_WINDOW_CLEAR_DEPTH = 0x02 }
Defines what should be cleared in a window. **Functions**

UGLIWindow UGLIWindowOpen (**UGLIMainContext** context, **UGLIWindowTag** tag,...)
void **UGLIWindowClose** (**UGLIWindow** window)
void **UGLIWindowRedraw** (void)
void **UGLIWindowSetSize** (int width, int height)
void **UGLIWindowSetTitle** (const char *title)
void **UGLIWindowSwapBuffers** (void)
void **UGLIWindowGetSize** (int *width, int *height)
int **UGLIWindowGetWidth** (void)
int **UGLIWindowGetHeight** (void)
void **UGLIWindowSetFullscreenViewport** (void)
void **UGLIWindowSetViewport** (int x, int y, int width, int height)
void **UGLIWindowSetFullscreenOrtho** (void)
UGLIEventHandleFlag UGLIWindowRunEventLoop (**UGLIPhysics** physics)
UGLIEventHandleFlag UGLIWindowHandleEvents (void)
void **UGLIWindowSendUserEvent** (**UserEventType** type, void *data1, void *data2)
void **UGLIWindowSavePng** (const char *filename)
void **UGLIWindowSetObject** (void *object)
void **UGLIWindowSetReshape** (void(*Reshape)(void *object, int width, int height))
void **UGLIWindowSetDrawContents** (void(*DrawContents)(void *object))
void **UGLIWindowSetEventHandler** (int(*HandleEvent)(void *object, const **UGLIEventPointer** e))
void **UGLIWindowSetClearColor** (const float *color)
int **UGLIWindowHandleEvent** (const **UGLIEventPointer** event)
void **UGLIWindowRequestRedraw** (void)
void **UGLIWindowSetMouseMotionAccept** (int accept)
void **UGLIWindowClear** (**UGLIWinClearFlag** flags)

Detailed Description

Window management.

Typedef Documentation

typedef struct UGLIWindowData* UGLIWindow

Window object.

Enumeration Type Documentation

enum UGLIWinClearFlag

Defines what should be cleared in a window.

Enumerator:

UGLI_F_WINDOW_CLEAR_COLOR Clear color buffer.
UGLI_F_WINDOW_CLEAR_DEPTH Clear depth buffer.

enum UGLIWindowTag

Tags that can be given to UGLIWindowOpen to give window specific properties.

Enumerator:

UGLI_T_WINDOW_DONE List end, no more tags. Should be always added after the last tag-value pair.
UGLI_T_WINDOW_XPOS Window X position, int. Default: System decides
UGLI_T_WINDOW_YPOS Window Y position, int. Default: System decides
UGLI_T_WINDOW_WIDTH Window width, int.
UGLI_T_WINDOW_HEIGHT Window height, int.
UGLI_T_WINDOW_TITLE Window title, char*. Default: NULL
UGLI_T_WINDOW_RESHAPE Reshape function, void (*Reshape)(void *object, int width, int height). Default: NULL
UGLI_T_WINDOW_DRAWCONTENTS DrawContents function, void(*DrawContents)(void *object). Default: NULL
UGLI_T_WINDOW_HANDLEEVENT HandleEvent function, int (*HandleEvent)(void *object, const UGLIEventPointer e). Default: NULL
UGLI_T_WINDOW_CLEARCOLOR Reshape function, float* (pointer to float[4]). Default: White

---

Function Documentation

void UGLIWindowClear (UGLIWinClearFlag *flags*)

Clears the specified parts (bitwise or) of the specified window.

Parameters:

flags	Which parts of window to clear.

void UGLIWindowClose (UGLIWindow *window*)

Closes specified window.

Parameters:

window	Window to close.

int UGLIWindowGetHeight (void)

Gets window height.

Returns:

Window height.

void UGLIWindowGetSize (int * *width*, int * *height*)

Gets window size.

Parameters:

width	Pointer to width variable.
height	Pointer to height variable.

int UGLIWindowGetWidth (void)

Gets window width.

Returns:

Window width.

int UGLIWindowHandleEvent (const UGLIEventPointer *event*)

Sends an event to window.

Parameters:

event	Event to send.

UGLIEventHandleFlag UGLIWindowHandleEvents (void)

Handles all events pending for the window & returns.

Returns:

event handle flags.

UGLIWindow UGLIWindowOpen (UGLIMainContext *context*, UGLIWindowTag *tag*, ...)

Opens window.

Parameters:

context	Main context to use.
tag	Sequence of UGLIWindowTag, value pairs followed by UGLIWIN_DONE. Currently only one window is supported, and further window open requests will fail.

new Window, if opened successfully (otherwise NULL)

void UGLIWindowRedraw (void)

Redraws contents of the current window & swaps buffers.

void UGLIWindowRequestRedraw (void)

Requests window redraw. Window will be drawn only once (no matter how many redraw requests were received) after handling all events.

UGLIEventHandleFlag UGLIWindowRunEventLoop (UGLIPhysics *physics*)

Runs window event loop with an optional physics handler.

Parameters:

Returns:

physics	Physics handler (or NULL for none).

event handle flags.

void UGLIWindowSavePng (const char * *filename*)

Saves the contents of the specified window into the specified file.

Parameters:

Returns:

filename	File to save to.

void UGLIWindowSendUserEvent (UserEventType *type*, void * *data1*, void * *data2*)

Sends an user event to the specified window.

Parameters:

type	User event type.
data1	Data pointer 1.
data2	Data pointer 2.

void UGLIWindowSetClearColor (const float * *color*)

Sets new window clear color.

Parameters:

color	A pointer to float[4] that specifies the new clear color (or NULL for default white).

void UGLIWindowSetDrawContents (void(*)(void *object) *DrawContents*)

Sets window DrawContents (a function that will be called whenever window needs to be redrawn).

Parameters:

DrawContents	New DrawContents function (or NULL for none).

void UGLIWindowSetEventHandler (int(*)(void *object, const UGLIEventPointer e) *HandleEvent*)

Sets window HandleEvent (a function that will handle incoming events).

Parameters:

HandleEvent	New HandleEvent function (or NULL for none).

void UGLIWindowSetFullscreenOrtho (void)

Sets the projection matrix to fullscreen orthogonal projection, so that OpenGL coordinates match the window pixels. Doesn't set the viewport. If you also want to set fullscreen viewport, call UGLIWindowSetFullscreenViewport aswell.

void UGLIWindowSetFullscreenViewport (void)

Sets window viewport to fullscreen.

void UGLIWindowSetMouseMotionAccept (int *accept*)

Sets whether or not the specified window accepts mouse motion events.

Parameters:

accept	Whether or not accept mouse motion events.

void UGLIWindowSetObject (void * *object*)

Sets window object (passed to Reshape, DrawContents & HandleEvent).

Parameters:

object	New object pointer (or NULL for none).

void UGLIWindowSetReshape (void(*)(void *object, int width, int height) *Reshape*)

Sets window Reshape (a function that will be called whenever window size changes).

Parameters:

Reshape	New Reshape function (or NULL for none).

void UGLIWindowSetSize (int *width*, int *height*)

Requests library to change window size. Actual size might be different from requested, and some backends might not even support changing resolutions at all.

Parameters:

width	Requested width for window.
height	Requested height for window.

void UGLIWindowSetTitle (const char * *title*)

Sets window title.

Parameters:

title	Title text (or NULL for none).

void UGLIWindowSetViewport (int *x*, int *y*, int *width*, int *height*)

Sets window viewport.

Parameters:

x	X offset of the viewport.
y	Y offset of the viewport.
width	Width of the viewport.
height	Height of the viewport.

void UGLIWindowSwapBuffers (void)

Swaps window buffers.