Bachelor’s thesis

Degree programme in Information and Communications Technology

2021

Joshua Kennedy

INTEGRATING A PHYSICS SIMULATION LIBRARY INTO ANIMATION SOFTWARE

BACHELOR’S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2021 | 35

Joshua Kennedy

INTEGRATING A PHYSICS SIMULATION LIBRARY INTO ANIMATION SOFTWARE

Click here to enter text.

Creating realistic motion that appears physically accurate in animation software can be a difficult and tedious process if done by hand. For this reason, many software packages include physics simulation functionality to aid in this process. This thesis aimed to deliver a case study on integrating a physics simulation library into an animation package called Bongo. In addition to providing a better understanding of physics integrations for games, this thesis aimed to explain a physics library's integration within the context of animation. This was achieved by first exploring problems with traditional animation techniques for creating physics-based animation. After that, PhysX and Bullet Physics, two popular physics libraries, were compared with Bullet chosen as the library to be integrated. Finally, the integration was completed, and the implementation details were discussed. The implementation details cover the full cycle of physics simulation from 3D scene conversion to eventual playback for the animator.

KEYWORDS:

Animation, Physics, Simulation, Bullet Physics, CAD

CONTENTS

LIST OF ABBREVIATIONS **5**

1 INTRODUCTION **6**

2 RHINOCEROS AND BONGO **8**

3 THE STRUCTURE OF BONGO ANIMATIONS **9**
3.1 Timeline 9
3.2 Keyframes 10
3.3 Tweening 11
3.4 Bongo User Data 12
3.5 Transformation Matrices 12

4 PHYSICS FOR ANIMATIONS **14**
4.1 What is a Physics Simulation? 14
4.2 Difficulties with Keyframes 16
4.3 Collision Detection 17

5 EVALUATION OF EXISTING PHYSICS LIBRARIES **18**
5.1 PhysX 18
5.2 Bullet 19
5.3 Comparison of Bullet and PhysX 19

6 INTEGRATION INTO BONGO **21**
6.1 Bullet World Creation 21
6.2 Rhinoceros 3D Document Conversion 22
6.3 Meshing of Rhinoceros Objects 22
6.4 Physics Properties 26
6.5 Adding Rigid Bodies 27
6.6 Time Stepping 28
6.7 Retrieving and Updating Transformations 30
6.8 Playback 30

7 CONCLUSION **32**
7.1 Results 32

7.2 Future Possibilities 33

REFERENCES 34

EQUATIONS

Equation 1. Calculation for the objects mass. 27
Equation 2. Calculation for the objects initial transform for the simulation. 28
Equation 3. Calculation for the physics timestep. 29
Equation 4. Calculation for the simulation's animation transform 30

PICTURES

Picture 1. Bongo's Timeline. 9
Picture 2. Bongo's Keyframe Editor. 10
Picture 3. Bongo's Curve Editor. 12
Picture 4. A concave shape (left) and convex shape (right). 15
Picture 5. A NURBS surface (near) and the equivalent mesh (far). 23
Picture 6. The corner of a rectangular shape and its margin in red. 25
Picture 7. The original shape in black, offset in cyan, and scaling operation in red. 26

PROGRAMS

Program 1. Registering the GImpact collision algorithm. 21
Program 2. Gathering object meshes for a physics simulation in Rhinoceros. 24
Program 3. Offsetting the collision mesh to hide the margin. 26
Program 4. Stepping the physics simulation. 29

LIST OF ABBREVIATIONS

AABB	Axis aligned bounding box
CAD	Computer aided design
IK	Inverse kinematics
NURBS	Non-uniform rational basis spline
SDK	Software development kit
UI	User interface

1 INTRODUCTION

Creating computer animations has and continues to be, a difficult and time-consuming process. Even with modern animation software suites, creating believable and realistic animations can still be challenging. Animators and software developers of animation suites are constantly looking for new tools to improve their workflow. Depending on the type of animation being created, it may be useful to automate the more time-consuming parts of the animation. One such example is integrating physics simulation software into an animation suite. Rather than resolving collisions and forces by hand, animators could instead use a physics library to do the calculations. This would dramatically increase the speed of iterating and creating the animation as well as provide more realistic results.

Bongo is an animation plugin for Rhinoceros 3D. For Bongo's third release, physics simulation was decided as a major feature. Many Bongo users simulate mechanical structures or have dynamic objects they would like to move naturally within scenes. Collision detection with a physics simulation is useful when analyzing these mechanisms to see where potential interference and bindings could occur. Physics also allows for quickly creating physically accurate motion. Simulations of gravity and other forces are very difficult to recreate accurately with keyframes. For these reasons, a physics simulation library that can handle these problems must be chosen and integrated.

Few theses are discussing the integration of physics libraries in general. "Real-time Physics Simulation" discusses creating a new physics simulation library (Parkkulainen 2019). This thesis aims to cover the creation of a new physics library for use in other applications but does not provide details of a final integration. "Developing a Game Engine with SDL" explains creating a game engine. This includes integrating an existing physics library for use (Hietala 2011). While it helps cover structural details when integrating the two, the underlying library, Box2D, is designed for two-dimensional graphics. It is also focused on game development and does not reflect the specific challenges animators may face. This thesis aims to bridge these gaps in knowledge by explaining the integration specifically for animation software.

First, the background details on Rhinoceros 3D and Bongo, software being worked on, will be discussed. Next, physics simulations will be explained and how they apply to the animation process. Following that, Bullet Physics and PhysX will be compared and the reasoning for choosing Bullet will be discussed. Next, the details of the integration will

be explained as well as problems and their solutions within Bongo and Rhinoceros 3D discussed.

2 RHINOCEROS AND BONGO

Rhinoceros 3D is a computer-aided design (CAD) application. Rhinoceros 3D was first released in 1998 and at the time of writing was in its seventh version. It has been used in a wide array of industries. Rhinoceros 3D has a variety of ways to design objects, but it is primarily a Non-Uniform Rational Basis Spline (NURBS) modeler. NURBS is a mathematical system for creating surfaces and curves (Rogers 2011). Subdivision surfaces have also been supported since Rhinoceros 7 (AEC Magazine 2021).

Rhinoceros 3D is also a platform for 3rd party development within its ecosystem through a system of add-on software known as plugins. Plugins for Rhinoceros 3D can be written in the C# and C++ programming languages (Belcher 2018a). Plugins interface with Rhinoceros through its software development kit (SDK). The SDK provides access to much of the data and functionality in Rhinoceros allowing developers to extend and improve the tooling. Rhinoceros also supports scripting through RhinoScript, a language specific to Rhinoceros based on VBScript (Fugier 2018b), as well as Python (Belcher 2018b). Scripting is usually used to automate tasks not natively supported by Rhinoceros 3D.

Bongo is an animation plugin for Rhinoceros. Bongo was written in C++ and was currently in its second release at the time of writing. Bongo extends the modeling functionality of Rhinoceros 3D with animation tools allowing designers to easily create visualizations for works created in Rhinoceros. Like Rhinoceros, Bongo has found use in a variety of fields. Bongo is a keyframe animator as opposed to a frame-by-frame animator. This means that known values are set at various times and are blended to create animations. This is more flexible than a frame-by-frame animation system since it allows animators to modify individual parameters rather than redraw series of frames when a change is desired. Bongo also makes use of the Rhinoceros Render Development Kit. This allows Bongo to interface with any third-party renderer plugin installed to Rhinoceros 3D and render the final animations with them.

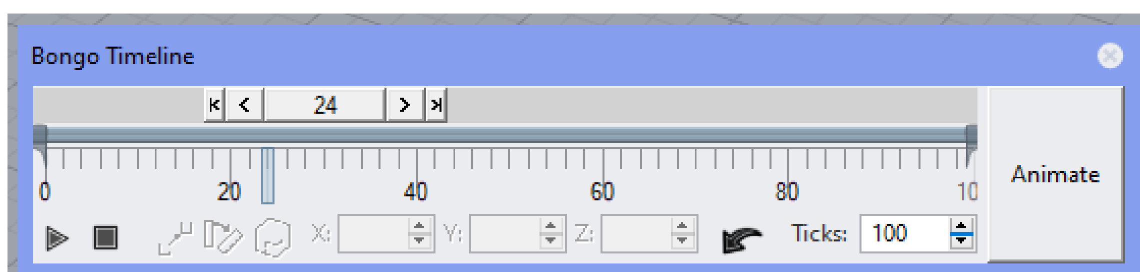
3 THE STRUCTURE OF BONGO ANIMATIONS

To understand how physics is integrated into Bongo, it must first be understood the structure of an animation. Bongo can animate many different properties from the motion of objects to their colors. To create and animate these values a variety of controls are used. These controls allow the animator a great deal of control when composing their work.

3.1 Timeline

Bongo's animations begin with the timeline. The timeline is a series of evenly spaced increasing values known as ticks (Robert McNeel & Associates 2010c). An animation can be any positive number of ticks in length. Animations are not required to begin at the zero tick or end at any specific point. These limits that define where the animation begins and ends are known as the animation limits. Ticks serve to discretely divide a continuous animation. Ticks do not correspond to any unit of time. Rather, the total animation length in seconds is set and the ticks subdivide it. Thus, as an animation grows and becomes more complex, an animator can extend and scale the ticks without disrupting the time scale of the animation.

The timeline (Picture 1) also serves a very important user interface (UI) purpose. It contains a slider that can be moved back and forth inside the animation (Robert McNeel & Associates 2010). Moving inside the animation is known as scrubbing the timeline. As the timeline is scrubbed animated values are updated to reflect their value at that tick. Scrubbing is used to preview the animation as the animator iterates through their design.

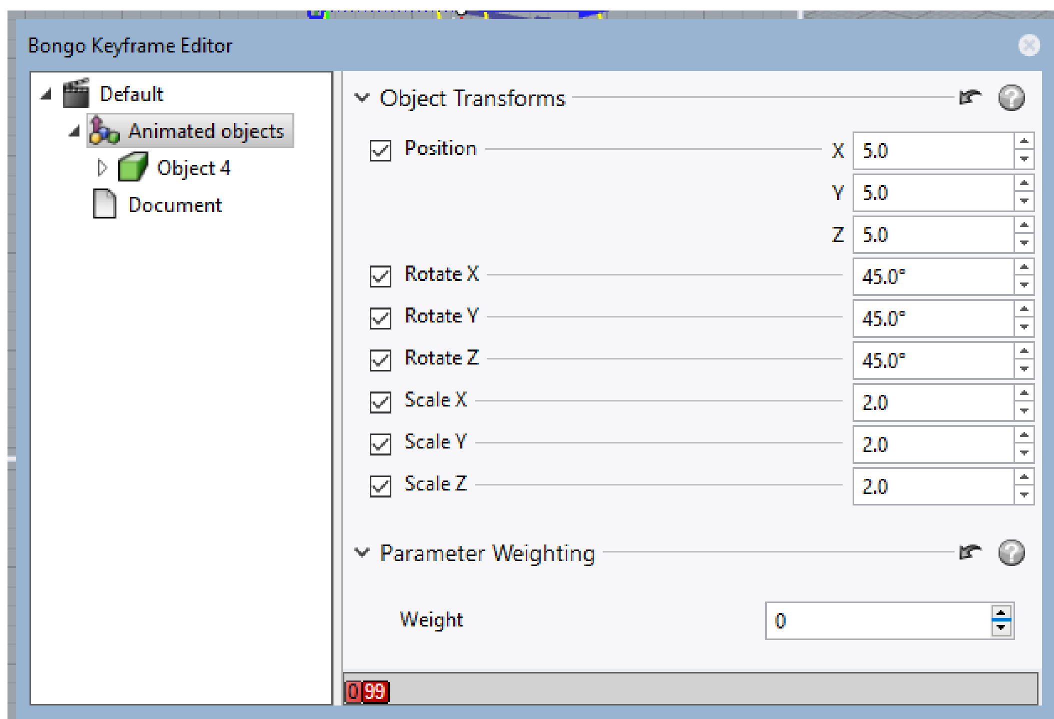


Picture 1. Bongo's Timeline.

3.2 Keyframes

Keyframes are known values in an animation at a given tick (Robert McNeel & Associates 2010b). For example, if an object's position is known to be at 5 meters in the Z direction at tick 25, a keyframe would be placed there. Keyframes in Bongo can hold a variety of types of values. These values can be vector or scalar quantities depending on which property of an object is animated. The most interesting properties that can be animated in a physics simulation are the object's position and rotation keyframes. These properties are vector quantities used to calculate the orientation of an object in three-dimensional space. The position X, Y, and Z values correspond to the X, Y, and Z positions in a Rhinoceros document. By animating these values an object is translated. The X, Y, and Z for an object’s rotation keyframes correspond to rotations about the X, Y, and Z-axis. When combined these will determine the orientation of an object in space.

Keyframes are created by modifying an object property in the user interface or adding them on the timeline. Once created, keyframes can be modified by entering the Bongo Keyframe Editor (Robert McNeel & Associates 2010) (Picture 2) and modifying the values there. While moving keyframes modifies the time at which the value is met, the keyframe editor modifies the value at that time.



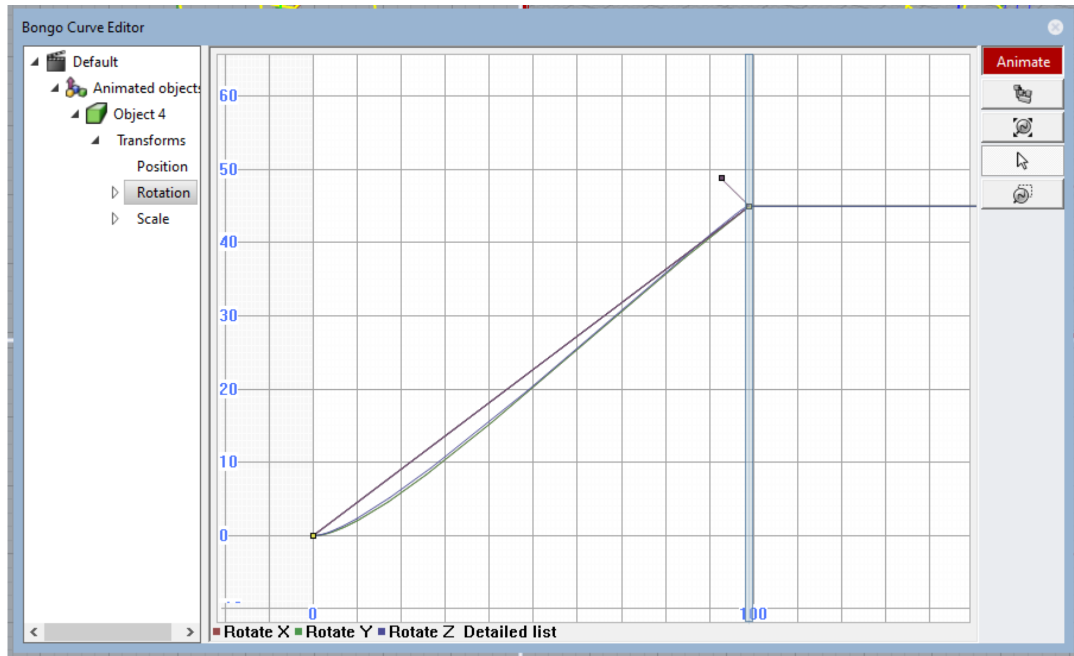
Picture 2. Bongo’s Keyframe Editor.

3.3 Tweening

To create a transition between two keyframed values, a blending function is needed. These blending functions are known as inbetweening, or more commonly tweening. Tweening creates a smooth transition from one known value to another by calculating the frames between keyframes (Robert McNeel & Associates 2010d). Bongo has several different types of tweening available. Tweening is useful because a simple linear interpolation between values is unnatural to the viewer.

Tweens can be adjusted to affect how the values are blended between. For a simple linear interpolation, the velocity of the changing value is constant. This does not create realistic animations as nothing naturally moves at a constant speed. Take for example a person walking from point A to point B. First, there is an acceleration phase as they get up to speed, then a constant velocity is maintained for the bulk of the journey, before finally slowing down as they arrive. Tweenings can help achieve this same effect.

Tweens can be adjusted inside the Bongo Curve Editor (Robert McNeel & Associates 2010a)(Picture 3). The curve editor contains a two-dimensional plot. The X-axis is time in the units of ticks for an animation. The value of the keyframe is the Y-axis. This creates a simple change-over-time graph. The first derivative of this graph is velocity. The derivatives can be adjusted using the curve handles allowing a user to modify how fast or slow an object leaves and arrives at its destination. This can help adjust the motion of an object, so it looks more realistic moving from one value to another. Adjusting the derivative of a curve is a useful tool but becomes tedious when many objects, with many properties, need to be modified.



Picture 3. Bongo's Curve Editor.

3.4 Bongo User Data

Since Bongo animates entities within Rhinoceros, it needs to interface with and modify them. It also needs to store its program data such as keyframes, tweening types, and animation settings. It accomplishes this by wrapping itself around the entities within Rhinoceros. Bongo can then store its information on the Rhinoceros entity through custom plugin data. This data is known as user data and is a feature of the Rhinoceros SDK (Baer 2018). Thus, by wrapping the entities in Rhinoceros, Bongo can save, load, and modify its information through a unified interface. Much of this data stored by Bongo are user settings. These settings are usually set by the user through the user interface, but others are inferred by Bongo.

3.5 Transformation Matrices

Rhinoceros, and by extension Bongo, change an object's disposition in space using a transformation matrix. A transformation matrix for a 3D system is a 4x4 matrix. A transformation matrix is capable of representing rotations, translations, scalings, shears,

among others. When an object's rotation or position is animated in Bongo these matrices are used to update the object in 3D space.

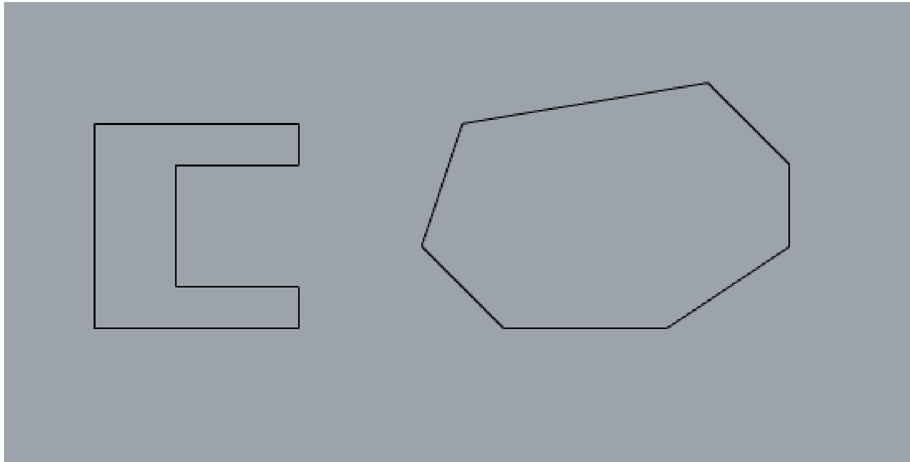
In many applications transformation matrices are used to give an object a coordinate system. The object coordinate system is the transformation from the world space to the object's current orientation in space (Shirley, Marschner 2009). Rhinoceros does not manage its objects this way. Instead, geometry exists in a world space frame and no orientation is managed for it. This makes animation complicated as knowing how and what to update is not so straightforward. Bongo resolves this by keeping track of an object transformation and an animation transformation. The object transformation is analogous to an object frame. The animation transformation keeps track of how much an object has been changed. When an object is updated in an animation these two transformations can be used to infer how an object needs to be transformed. These matrices will also become important when creating and updating the physics simulation.

4 PHYSICS FOR ANIMATIONS

4.1 What is a Physics Simulation?

A physics simulation is a recreation of a real-world occurrence within a virtual environment (Ryoo, Kim 2015). The software library that runs a physics simulation can also be known as a physics engine. There are a variety of physics libraries including Bullet Physics, NVIDIA PhysX, and Havok. Physics simulations can simulate a variety of things but many primarily simulate rigid and soft body dynamics. Dynamics is the study of the motion of objects and the factors that affect them such as mass, forces, and momentum (Britannica, T. Editors of Encyclopaedia 2007). Rigid bodies are objects that cannot be deformed like steel beams. A soft body is an object which is capable of deformation like a piece of cloth.

To simulate the dynamics of an object, they must have some shape or form that can be simulated. This shape can more generally be referred to as a physics shape. This shape is used to determine whether or collisions have occurred. Most physics libraries can simulate primitive shapes such as boxes, cones, spheres, and capsules as well as shapes based on meshes. Meshes are a collection of vertices and polygons that create a three-dimensional shape (Mukundan 2012). One way to describe meshes is as concave and convex. An example of a convex and concave polygon is given (Picture 4). Concave shapes curve inwards whereas convex do not. Concave shapes are generally more difficult to simulate so some physics libraries may not support calculating their dynamics.



Picture 4. A concave shape (left) and a convex shape (right).

For an object to be simulated the physical properties of the object must first be known. These physical properties correspond to real-world properties an object will have. These include friction coefficients, mass, among others. By modifying these parameters of an object in a simulation the results can be altered to represent a real-world object more accurately.

Physics simulations, like the animation or game they are running in, occur over time. The physics simulation, unlike real life, is not continuous. It physics simulations advance time in a series of discrete steps (van den Bergen, Gregorius 2010). The length of the step is used to update the position of an object based on velocity, mass, among others. The process of advancing the simulation is known as stepping the simulation. Most physics engines can use a variable timestep to stay in sync with whatever application it is running inside.

As previously mentioned, the simulation takes place over a series of discrete steps. It is important to understand what is happening at each step. A physics simulation usually begins with the application of forces. These may originate internally, as is the case with gravity, or set from outside the simulation. With the forces resolved the next step is determining velocity. Velocity allows the simulation to estimate the next position of the object. Once object positions are estimated the simulation carries out broad phase collision detection. Broad phase collision detection is an optimized collision detection algorithm that finds likely pairs of colliding objects (LaValle 2019). The broad phase does not know for certain if they are colliding or not but instead uses some metric to determine whether they are or not. Once the broad phase has found collision pairs they are sent to the narrow phase collision detection system. Narrow phase collision detection uses much

more computationally expensive collision detection algorithms to determine whether an object is colliding or not (LaValle 2019). Since they are more computationally it is prudent to calculate them as few times as possible. Therefore, the broad phase collision detection step is necessary. Once collisions are found they are resolved, and the final forces and positions can be updated for the next step.

Physics simulations tend to separate objects in two ways, dynamic and kinematic. Dynamic objects are those which are subject to the rules of the simulation. These feel the various forces from collisions and gravity. Kinematics, unlike dynamics, is motion without regard to the forces involved (Britannica, T. Editors of Encyclopaedia 2017). For this reason, kinematic objects do not follow the simulation rules. Their motion is derived from outside the simulation. For a game, this could be a character that moves around from player control, or a vehicle moving with keyframes in an animation. While a vehicle or human is subject to the rules of a physics simulation, it is much simpler to model them as a kinematic object.

4.2 Difficulties with Keyframes

Keyframing motion for objects in animation is already a time-consuming process. If an animator would attempt to keyframe the motion of an object from gravity or other forces, and accurately respond to collisions the time investment would grow rapidly. Furthermore, to appear correct, all keyframes must be at the mathematically correct values to be convincing to the viewer. Currently, the only way for an object to be at the mathematically correct position in Bongo is to do the calculation by hand. Also, there is very rarely a single object moving under physical forces in a scene. For each object, the process must be repeated when an animator’s time could better be spent elsewhere. The animation also then becomes very difficult to change. Any modification, even very small, has a cascading effect that requires changes throughout the rest of the animation. It would be much easier for the animator to define the starting conditions of a physics simulation and modify those objects which interact in it. Then the simulation parameters can be iterated on, instead of the objects' keyframes themselves, allowing the animator to prototype much faster.

4.3 Collision Detection

One problem an animator would want to avoid is collisions between objects. In a normal keyframed animation in Bongo, nothing prevents objects from overlapping or intersecting one another. While Rhinoceros 3D has tools for finding unions and intersections between geometry, they are not applied constantly while animating. This can cause the unwanted side effect where objects appear to move through one another. Physics simulation allows animators to automatically resolve and prevent collisions.

Collision detection is also important for animators analyzing the motion of mechanisms they design in Rhinoceros 3D. Many users of Bongo take advantage of its inverse kinematics (IK) engine to simulate mechanisms such as a door hinge. Collision detection can help find potential bindings and lockups within these mechanisms.

5 EVALUATION OF EXISTING PHYSICS LIBRARIES

Before development work could begin on integrating a physics library into Bongo, a suitable library needed to be found. While several physics libraries exist, the candidates were narrowed down to Bullet Physics and NVIDIA PhysX. Both were chosen because of their permissive licenses and widespread use.

5.1 PhysX

PhysX is an open-source real-time physics engine created by NVIDIA. It is integrated into several existing applications including the Unity and Unreal game engines. PhysX is written in C++ and uses the CMake build system for generating projects (NVIDIA Corporation 2018).

PhysX is optimized for multithreaded simulations as well as those able to run on the graphics processing unit. It supports mesh collision detection as well as primitive shapes. Concave meshes are supported only if static in the physics scene. If meshes are intended to be dynamic, they must use a convex collision shape. PhysX also natively supports different units and scales (Nvidia Corporation 2018).

PhysX is copyrighted by NVIDIA Corporation and licensed under the BSD-3 software license. BSD-3 is a permissive license allowing for unlimited redistribution so long as the copyright and disclaimers are maintained. This allows for PhysX to be used in commercial applications.

PhysX is currently in its fourth version which was released in December of 2018. Its successor, PhysX 5, has been announced but has not been released yet. Since it is open-source, it allows developers to investigate its inner working to identify and resolve bugs if the need arises. Large new features are often only integrated by NVIDIA in between major versions. If PhysX does not support a required feature, it is unlikely to be resolved in that version. On the other hand, since it is developed by a large hardware vendor and used in a variety of applications it is quite stable.

5.2 Bullet Physics

Bullet Physics is an open-source real-time physics library created by Erwin Coumans. Since its release, it has been used in a variety of applications from robotics to games. It has developed a strong open source following who feed improvements back into the library. Like PhysX, Bullet is also written in C++ and uses the CMake build system (Coumans 2020). Besides the C++ interface Bullet also supports a Python library known as PyBullet.

Bullet supports collision detection between primitive shapes, static mesh shapes, as well as dynamic convex and concave mesh shapes. Bullet Physics is also capable of rigid and soft body dynamics simulation. Furthermore, it has specialized simulations for multibody dynamics and can calculate inverse dynamics. Bullet also comes packaged with a variety of examples and third-party extensions. These extensions include convex decomposition tools, inverse kinematics solvers, and tools for serializing and deserializing physics simulations (Coumans 2015).

Bullet Physics is licensed under the Zlib software license. Zlib is a permissive software license that allows unlimited distribution under restrictions. These restrictions require that the authors of the original software must not be misrepresented, and the license must be distributed with the source code.

Since Bullet is actively maintained by Erwin Coumans and receives development from the community, it is updated frequently. Major features are added by Erwin and the community regularly. It also has a large developer community around it which can help resolve issues should they arise. By in large, Bullet is quite stable. The oldest portions of the codebase have been used by a variety of software projects successfully. Newer, and more experimental features may not be up to the same standard.

5.3 Comparison of Bullet and PhysX

Bullet and PhysX both share many features. They are both capable of simulating rigid body dynamics and detecting collisions between a variety of shapes. Where Bullet differs is its ability to simulate dynamic concave shapes. Dynamic shapes can move and interact in the simulation. A workaround for dynamic concave shapes in PhysX would be a process of convex decomposition. Convex decomposition breaks down a concave shape

into a set of convex pieces (Lien, Amato 2008). There are existing libraries available for use that are capable of convex decomposition (Mammou 2012), but it is an inferior solution to native support.

Bullet Physics was written with meters as the unit for simulations. PhysX natively supports multiple unit systems. Rhinoceros 3D allows users to model in a variety of unit systems. For these various unit systems forces, masses, and shapes must be scaled to work correctly in Bullet. PhysX on the other hand can natively support these. While a compelling feature for Bongo, a workaround exists within Rhinoceros. The Rhinoceros SDK allows for easily converting between unit systems through the *ON_UnitSystem* class.

The biggest concern when choosing a physics library was ensuring that the library could simulate the types of objects Bongo users are creating. Since many users are animating objects that may eventually be manufactured, they often do not have perfectly convex shapes. These parts may have screw holes or be gears in an assembly. A convex shape would simply be inadequate for a physics simulation. While convex decomposition could be done in both PhysX and Bullet, integrating additional software libraries for this task greatly increases project complexity. Native support for concave shapes without external libraries was provided by Bullet. For this reason, Bullet was chosen as the library to be integrated into Bongo.

6 INTEGRATION INTO BONGO

A physics simulation for Bongo is initiated in one of two ways. The first is the animator invokes the simulation via a Rhinoceros command. The second is an automatic solver. When a change is detected in the animation the physics simulation is flagged as needing calculation. The next time the user scrubs the timeline the physics simulation will calculate as well. Once one of these signals has been received Bongo can begin the necessary steps of using Bullet to calculate a physics simulation.

6.1 Bullet World Creation

Bullet requires several steps to set up before a simulation can be computed. The first step is creating the physics world. The physics world is the topmost interface for building and updating the simulation. Bullet has a variety of physics worlds, each specialized to the application. For rigid body simulation, the *btDiscreteDynamicsWorld* is adequate. The *btDiscreteDynamicsWorld* has several dependencies that need to be created before the world can be used.

The first step for creating a physics world is the collision configuration. The collision configuration is responsible for memory management within Bullet as the simulation runs. It was found that the *btDefaultCollisionConfiguration* is satisfactory. The second prerequisite is the broad phase collision detection algorithm. Bullet Physics comes with a variety of broad phase collision detection algorithms, but the *btDbvtBroadphase* was determined to be adequate. The third is the collision dispatcher. The collision dispatcher is Bullet's narrow phase collision detection algorithm. Like the broad phase, algorithm Bullet comes with several different types of collision dispatchers. The *btCollisionDispatcher* proved adequate for Bongo's requirements.

By default, Bullet does not take advantage of the concave collision algorithms. For most gaming applications convex collision detection is suitable. The concave collision algorithm also is only available through a specialized interface within Bullet. To make use of it the one must register the algorithm with the *btCollisionDispatcher.*

Program 1. Registering the GImpact collision algorithm.

```
btGImpactCollisionAlgorithm::registerAlgorithm(m_pDispatcher);
```

Finally, the physics world can be created. The physics world is constructed with the prerequisites passed in as parameters to the constructor.

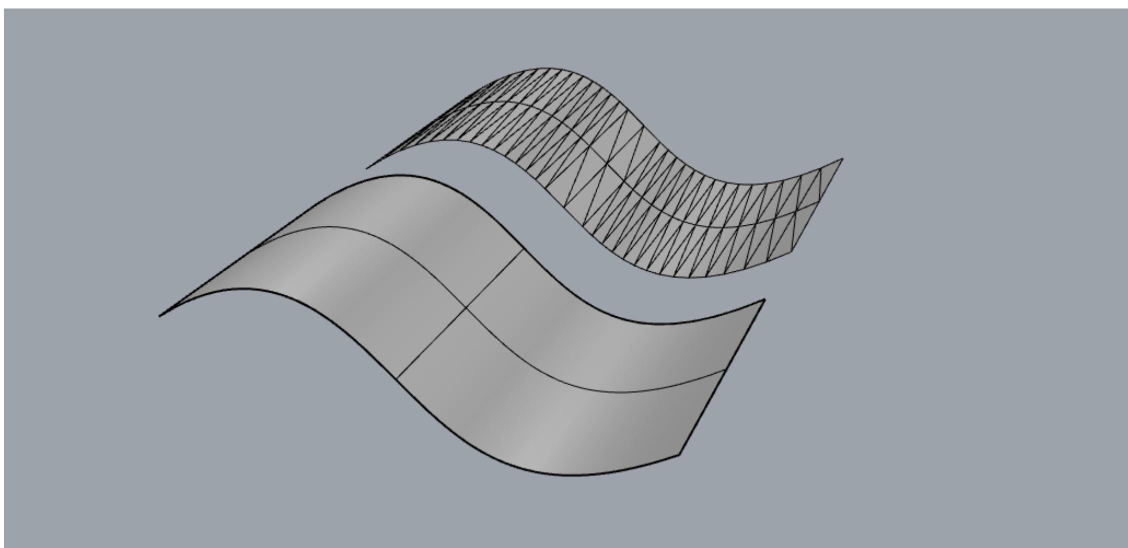
6.2 Rhinoceros 3D Document Conversion

All the items inside a Rhinoceros 3D file are contained in a structure known as the document. These items can be objects, which are generally geometry created or imported into Rhino, lights, and materials for rendering, among others. To create a physics simulation, objects eligible for a simulation must first be found.

Objects are stored in the Rhinoceros document on the object table. The object table is traversed with the *CRhinoObjectIterator* (Fugier 2018a). The iterator allows filters to be set which can automatically skip over objects that are not eligible for a simulation. For Bongo, only objects with some geometries are desired. These shapes are the previously mentioned SubD and NURBS geometry. To do this, properties are set on the iterator to exclude lights and grips. Lights have no geometric shape, and grips are a Rhinoceros tool for modifying geometric objects. From there iterating begins. For each object it is checked if it is animated in Bongo, then if its animation is enabled, and finally if the object has physics enabled. All this data is accessed through the Bongo SDK and is stored in the Bongo user data. When an eligible object is found, it is meshed before passing it on to be added to the simulation.

6.3 Meshing of Rhinoceros Objects

Bullet handles collisions using primitive shapes and meshes. While Rhinoceros 3D supports meshes as a geometry type, it is not the primary way users create objects. For this, NURBS surfaces or subdivision surfaces are used. To use these objects in Bullet they first need to be transformed to mesh. This process is known as meshing. Meshing tessellates, or splits a surface, into many smaller surfaces. An example of meshing is shown (Picture 5). In meshing these smaller surfaces are usually a series of triangles.



Picture 5. A NURBS surface (near) and the equivalent mesh (far).

Rhinoceros 3D has methods for retrieving a variety of different meshes from objects. The render mesh is used in Rhinoceros's display. These are the meshes that are ultimately drawn to the screen in the display and used for renderings. For physics simulation, a special version of the render mesh has chosen to be extracted. More complex meshes take more time to calculate collisions for in Bullet. Because of this other, more accurate mesh types may not be suitable.

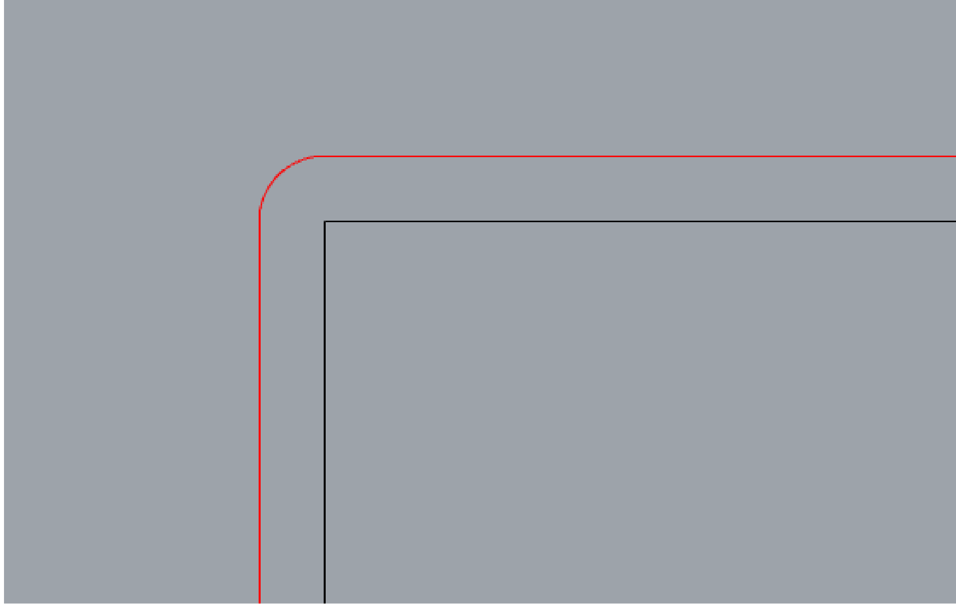
Bullet has some limitations on meshes when they are created for the simulation. The largest concern is the edge length. Bullet suggests edges no longer 5 units in length (Coumans 2015). This is because larger edge lengths become numerically unstable when calculating collisions. Fortunately, the Rhinoceros SDK allows tuning this through the *ON_MeshParameters*. By defining a maximum edge length in the *ON_MeshParameters* Rhinoceroses mesh creation system can be modified to match Bullets needs.

Program 2. Gathering object meshes for a physics simulation in Rhinoceros.

```
CRhinoObject* pObject = /*Gather object here*/;
ON_MeshParameters MeshParameter = ON_MeshParameters();
pObject->GetRenderMeshParameters(MeshParameters);
MeshParameters.SetMaximumEdgeLength(5.0);
pObject.CreateMeshes(ON::render_mesh, MeshParameters);
ON_SimpleArray<const ON_Mesh*> aMeshes = ON_SimpleArray<const ON_Mesh*>();
obj.GetMeshes(ON::render_mesh, aMeshes);
```

Not all polygons handled in Rhinoceros are the same. Primarily they are triangles and quadrilaterals. Rhinoceros also has support for polygons with any number of sides. The polygons with an arbitrary number of sides are more commonly known as *ngons,* but these are not created when meshing. When creating concave shapes for Bullet, they must be triangular meshes. This means that every face polygon is a triangle. To meet this requirement, the meshes must be triangulated through the *ON_Mesh::ConvertQuadsToTriangles* function.

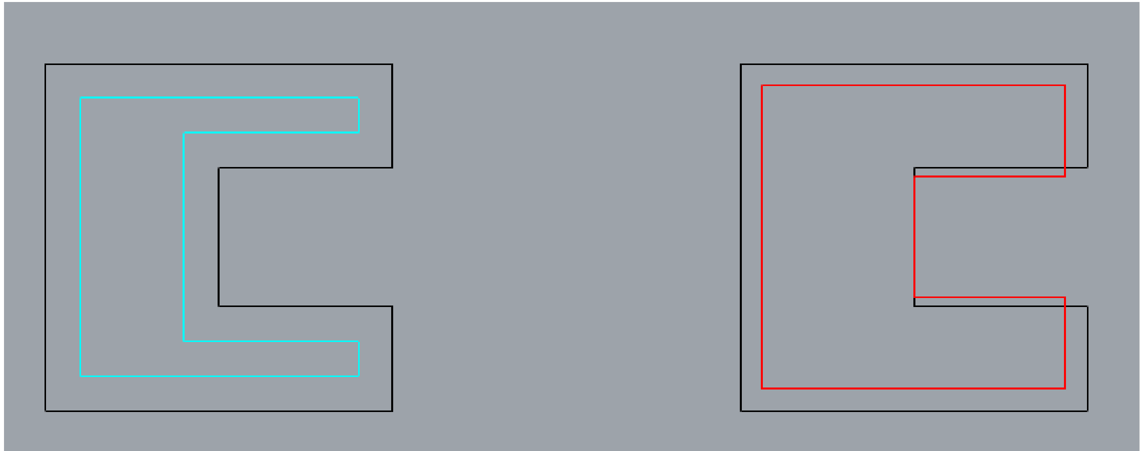
Bullet adds a small margin around objects to help keep simulations stable. When collisions occur with penetrations, meaning one object overlaps another, simulation performance and reliability may suffer (Coumans 2015). This margin is padding which slightly enlarges the object's shape so when objects become close enough a collision can be said to occur (Picture 6). This margin helps keep simulation stable as undoing penetrations are difficult to accurately compute. It is better to use the margin to resolve the collision at a point of a near hit.



Picture 6. The corner of a rectangular shape and its margin in red.

When testers first began using Bongo 3 these margins were found to be undesirable. For games, the collision margin is often unnoticed as it is so small, and things move so quickly a player is unlikely to notice it. Animators when reviewing the simulation would notice these gaps and how they impacted animation realism, especially on mechanisms where a camera is likely to be zoomed in close to the various linkages. Margins are a user-adjustable parameter in Bullet, so it was first attempted to set the margins to zero. This was found to be inadequate since the simulation's accuracy began to suffer.

In Bongo, it was decided to offset the mesh by the amount of the margin. This is superior to a scaling operation the distance of the margin since scaling changes the structure of the shape around the concave portion (Picture 7). An offset moves the vertices in a mesh back along the meshes vertex normal. A vertex normal is simply the direction a vertex is facing. Vertex normals are primarily used in graphics for lighting calculations but are also quite useful here. By offsetting the mesh the distance of the margin, it can be effectively hidden while maintaining the original shape of the concave object.



Picture 7. An example of the original shape (black), resulting shape after an offset (cyan), and resulting shape after being scale (red).

Program 3. Offsetting the collision mesh to hide the margin.

```
#define BULLET_DEFAULT_MARGIN_OFFSET 0.04
ON_Mesh* pOffsetMesh = RhinoMeshOffset(BULLET_DEFAULT_MARGIN_OFFSET, pMesh);
```

6.4 Physics Properties

Bullet provides an interface for setting physics properties so that objects behave more similarly to their real-world counterparts. The first property, and probably most important is an object's mass. This can be any positive number. The unit for mass in Bullet is kilograms. Then come two friction coefficients, one for static friction and the other for rolling. Both coefficients are in the range of zero to one. Finally, there is the object's coefficient of restitution. Like the friction coefficients, they are also clamped between zero and one. The coefficient of restitution is the coefficient of the velocity of the object before the collision divided by the velocity after the collision. In simpler terms, this means how much velocity is retained by an object through a collision. A value of one corresponds to a perfectly elastic collision.

In Bongo, it was decided on two pathways to determine the object's physical properties. The first is having the settings manually entered by the user through the Bongo Object Properties panel. The Bongo Object Properties panel is a user interface for setting various animation properties on an object. The second is the properties are inferred from the objects render material. The render material is the set of properties that determine

how an object will look when drawn on the screen. These include color, textures, and other values that affect appearance. In Rhinoceros 3D several predesigned materials correspond to real-life materials. These real-life materials are gem, glass, metal, paint, plaster, and plastic. From the properties of the real-life materials, the corresponding simulation parameters can be inferred.

Since there are only a small set of render materials that correspond to real-world objects a lookup table can be created which will contain the properties. When the simulation begins, instead of taking the parameters from the user data the lookup table can be consulted. This table will hold the materials density, friction coefficients, and restitution coefficient. The coefficients can simply be set on the objects when they are created. By also storing the density of the material it can be used to calculate the mass. To do this, an object's volume must first be known. The volume can be calculated through the *ON_Mesh::Volume* function which returns the volume in document units cubed. By converting the volume to meters cubed the calculation of the object's mass can be found using the density formula.

$$M = V * D$$
$$M = mass$$
$$V = volume$$
$$D = density$$

Equation 1. The calculation for the mass of the object.

6.5 Adding Rigid Bodies

Now that the object's physics mesh and physics properties are known, objects can begin being added to the simulation. Since the mesh has been previously calculated, it can be used to create the physics shape. Like many other Bullet features, there are several different supported shape types. Primitive shapes, such as boxes, spheres, and convex hulls, are supported but are inadequate for our needs. Recalling from previous sections that the simulation intends to use concave mesh shapes. For this reason, the *btGImpactMeshShape* was chosen. The *btGImpactMeshShape* is the specialized shape object in Bullet capable of representing a concave shape. To create the mesh shape, the Rhinoceros mesh data must be copied into a *btTriangleIndexVertexArray.* The

btTriangleIndexVertexArray is passed as an argument to the *btGImpactMeshShape.* Finally, to make sure the shape is properly configured based on the mesh information for the simulation, btGImpactMeshShape::*updateBound* must be called.

With a physics shape created the corresponding rigid body object for the simulation can be created. Rigid body objects in Bullet are created with the *btRigidBody* class. To create a rigid body a physics shape must be supplied. In this case, the shape is the *btGImpactMeshShape.* The physics properties previously determined in the Physics Properties section can be set through the corresponding interface on the *btRigidBody*. Kinematics objects are user-defined with the value stored in Bongo's user data. To set a rigid body as kinematic within Bullet, the object's mass property is set to zero. Dynamic objects have a mass set from the physics properties.

When setting up the simulation in Bongo, the starting orientation for objects needs to be known. This is the object's initial position and rotation. Recalling from section 3.5 that Bongo objects have an animation and object transformation. The initial transformation can be computed by multiplying the inverse of the animation transform with the current object transform.

$$InitialTransform = AnimationTransform^{-1} * ObjectTransform$$

Equation 2. The calculation for the object's initial transform for the simulation.

With the initial transform, the rigid body transformation can be set in Bullet. Since both Bullet and Rhinoceros represent these as 4x4 transformation matrices it is a simple copy of the values from one to the other. With the initial transformation set, the object is then added to the physics world.

6.6 Time Stepping

Bullet Physics runs its simulations in a series of discrete steps. An individual step is known to Bullet as a time step. A time step corresponds to how much time has passed in the simulation in seconds. The time step is used to apply forces and move objects. Bullet Physics was originally designed as a game's physics engine. The amount of processing occurring in a game can vary heavily depending on what is currently happening in a game. Because of this, the physics simulation for a game will often run

at a variable frequency. Therefore, Bullet takes a time step for the simulation stepping instead of frequency. A variable framerate is undesirable for animation use. Animators desire predictable outcomes, and a shift in the frequency can cause unforeseen consequences in the resultant animation. Therefore, the simulation stepping is presented to the user as a unit of frequency.

The frequency at which the simulation is updated can have a large effect on the simulation's results. Bongo, by default, uses a simulation frequency of 100 hertz. This is stored in the Bongo user data. The timestep would then correspond to the period of the simulation frequency. It can be easily calculated.

$$T = \frac{1}{f}$$

$$T = time$$

$$f = frequency$$

Equation 3. The calculation of physics timestep.

This is then used to step the Bullet simulation.

Program 4. Stepping the physics simulation.

```
m_pDynamicsWorld->stepSimulation(dTimeStep);
```

The simulation frequency needs to be tuned to yield convincing simulation results. Take for example a millimeter-scale model. Gravity acceleration is a constant -9.81 meters per second squared in the Z direction. After one second an object can be moving 9.81 meters per second. This would correspond to 9,810 units inside the Rhinoceros document per second or 9.81 millimeters per time step at 100 hertz. A small solid object, less than 5 millimeters thick, may pass through a thin object because it would not have intersected when transformations were first predicted.

Originally, the simulation frequency was based on the unit system the model was created in. This was done by using a higher frequency at smaller unit scales. This was so smaller time steps would be used and it was more likely collisions between small objects would not be missed. This had the side effect of making the simulation take much longer to compute. Users at larger scales also would have complex scenes, and a smaller timestep can help achieve more accurate simulation results. For these reasons, the simulation frequency is a user-adjustable parameter.

6.7 Retrieving and Updating Transformations

As a simulation iterates over the number of timesteps, the orientation of rigid bodies is updated inside the dynamic's world. These transforms can be retrieved through a specialized structure in Bullet known as a *btMotionState*. The motion state is used to interpolate motion when its values are set for kinematic objects and retrieve the object's transformation when it is a dynamic rigid body.

Retrieving a transform from a dynamic rigid body is easy but saving it in a state for Bongo to use later is less straightforward. Recall from the Adding Rigid bodies section that the initial transform from Bongo objects has been calculated. When Bongo transforms objects, it is moving and reorienting the object from where it currently is. When retrieving the transforms from Bullet they must be stored for Bongo in the same way. To do this the current object's position is found and the change from the beginning is computed. This is done by multiplying the current world space transform by the inverse of the initial transform.

$$Change = Current * InitialTransform^{-1}$$

Equation 4. The calculation of the simulation's animation transform

Once the change is found it is cached for playback later. The cache is an internal structure for Bongo. It holds the beginning and ending ticks of the simulation and all the transformations in sequence.

For updating kinematic rigid bodies, the animation transformation is retrieved from the Bongo object interface at that time. Since this transform is a change, and not a world space transform, the previously discussed process is reversed. This value is then used to update the *btMotionState* with the result.

6.8 Playback

Once a simulation has been computed the results need to be played back for the animator. Since the transforms are cached, they can be retrieved for playback whenever the animation time has changed. Unfortunately, fetching the closest cached transform at a tick is not enough for animators. Like keyframes and tweening, a value is expected to

be defined for every point of time in the animation. As a result, the transforms must be interpolated.

Recall from section 3.5 that object transformations are represented as 4x4 transformation matrices. These matrices can be broken down into several simpler transformations. This process is known as decomposing a transformation matrix. Transform matrix decomposition is a well-studied problem (Thomas 1991). Most matrix decompositions algorithms first remove the translation, as a three-dimensional vector corresponding to a point in space, then the three scalars corresponding to an object's scaling along the X, Y, Z, and finally the rotation as Tait-Bryan angles. Tait-Bryan angles correspond to three rotations about the X, Y, and Z-axis, respectively. Tait-Bryan angles are not useful for interpolation when extracted from a rotation matrix. This is because orientations that are near each other may not extract as similar angles. When these dissimilar angles are interpolated it may cause large, undesirable rotations to occur. For this reason, the rotations are extracted as quaternions. Quaternions, like Tait-Bryan angles, can be easily extracted from a transformation matrix (Shoemake 1991). Quaternions are also able to easily be interpolated.

To get the correct transform at any timestep it must first be determined which two cached physics timesteps the desired time lies between. From there, it is calculated how far between the two timesteps the desired time is. This will tell us how far along the values need to be interpolated. Next, the cached transforms are decomposed. These give simple transforms that can easily be interpolated between. Finally, the transformation is recomposed from the simpler transforms, and the objects are updated inside of Rhinoceros.

7 CONCLUSION

In this thesis, a case study was presented on integrating a physics library for use in animation software. Animation software presents its own unique set of challenges for a developer when carrying out the integration. Much of the existing knowledge also focuses on physics for games. By focusing on animation, problems not encountered before can be identified and remedied both at the library and integration levels.

7.1 Results

When integrating a physics library into Bongo, problems faced by animators were first identified and examined. These problems differ from issues players may face when experiencing game physics, and thus require specific solutions. Two popular physics libraries, Bullet Physics and PhysX were then compared through the lens of these problems. As a result, Bullet Physics was found to be best suited for integration into animation software. These results can help future developers more rapidly conclude whether a specific physics library is suitable for their needs.

Next, the integration was carried out into Bongo. The integration covered details from the conversion of the 3D scene, retrieving and updating transforms, as well as eventual playback. These details are particularly useful for future developers when carrying out their integration since many of these details are not specific to any one physics library. Specific solutions for problems within Bullet Physics and the Rhinoceros 3D SDK are also given contributing to greater knowledge of how to utilize these feature-rich tools. As testers reviewed the integration, additional issues were found. These issues, such as margins and a consistent simulation frequency, were then remediated. These problems are specific to a physics library integration for animation and thus increase the collective knowledge within this domain.

As a result of the integration, the capabilities of animators within Bongo have greatly been extended. Animators can now create much more convincing and realistic animations based on physics. These greatly increase the visualization possibilities of Bongo as well and can speed up the delivery of complex scenes. The features also allow new types of animations to be created in Bongo that may not have previously been possible.

7.2 Future Possibilities

In the future, the physics integration for Bongo could be deepened by taking utilizing more of the features Bullet has available. These include simulations such as soft body physics objects and constraints. Constraints would allow users the ability to create mechanical structures whose motion could be simulated with forces. The standard methodology for doing this currently within Bongo is through the IK system. Soft body simulations are useful for objects like cloth and rope. Certain industries, such as jewelry designers may find this useful. Support for the primitive collision shapes could also be added to allow the user finer control over how objects behave in the simulation. Primitive collision shapes are also much more computationally efficient. The ability to iterate the simulation faster with these shapes could be very useful to animators.

Another possible extension is creating a generic interface for different physics simulation libraries in Bongo. Currently the integration is built specifically for Bullet Physics. With a common interface for different physics libraries support for different features could be added. This would allow users to select different physics libraries based on their animation needs.

REFERENCES

AEC MAGAZINE, February 1, 2021-last update, The Rhino 7 interview: SubD + Rhino.Inside.Revit [Homepage of AEC Magazine], [Online]. Available at: https://aecmag.com/cad/the-rhino-7-interview/ [May 5, 2021].

BAER, S., December 5, 2018-last update, Plugin User Data. Available at: https://developer.rhino3d.com/guides/rhinocommon/plugin-user-data/ [April 8, 2021].

BELCHER, D., December 5, 2018a-last update, What is a Rhino Plugin?. Available at: https://developer.rhino3d.com/guides/general/what-is-a-rhino-plugin/ [April 30, 2021].

BELCHER, D., Dec 5, 2018b-last update, What is Rhino.Python?. Available at: https://developer.rhino3d.com/guides/rhinopython/what-is-rhinopython/ [April 30, 2021].

BRITANNICA, T. EDITORS OF ENCYCLOPAEDIA, June 21, 2017-last update, Encyclopedia Britannica "Kinematics". Available at: https://www.britannica.com/science/kinematics [April 25, 2021].

BRITANNICA, T. EDITORS OF ENCYCLOPAEDIA, Feb 7, 2007-last update, Encyclopedia Britannica "Dynamics". Available at: https://www.britannica.com/science/dynamics-physics [April 19, 2021].

COUMANS, E., 2020. *Bullet Physics.* https://github.com/bulletphysics/bullet3: .

COUMANS, E., 2015-last update, Bullet User Manual. Available at: https://github.com/bulletphysics/bullet3/blob/93be7e644024e92df13b454a4a0b0fcd02b21b10/docs/Bullet_User_Manual.pdf [April 5, 2021].

FUGIER, D., Dec 5, 2018a-last update, Iterating the Geometry Table. Available at: https://developer.rhino3d.com/guides/cpp/iterating-geometry-table/ [April 30, 2021].

FUGIER, D., Dec 5, 2018b-last update, What are VBScript and RhinoScript?. Available at: https://developer.rhino3d.com/guides/rhinoscript/what-are-vbscript-rhinoscript/ [April 30, 2021].

HIETALA, O., 2011. *Developing a Game Engine with SDL*, Kajaanin ammattikorkeakoulu.

LAVALLE, S.M., 2019. *Virtual Reality.* Cambridge University Press.

LIEN, J. and AMATO, N., 2008. Approximate convex decomposition of polyhedra and its applications. *Computer Aided Geometric Design,* **25**, pp. 503-522.

MAMMOU, K., 2012. *Volumetric Hierarchical Approximate Convex Decomposition.* https://github.com/kmammou/v-hacd: .

MUKUNDAN, R., 2012. *Advanced Methods in Computer Graphics: With Examples in OpenGL.* Springer.

NVIDIA CORPORATION, 2018. *PhysX.* https://github.com/NVIDIAGameWorks/PhysX: .

NVIDIA CORPORATION, December 21, 2018-last update, PhysX 4.0 SDK Guide. Available at: https://gameworksdocs.nvidia.com/PhysX/4.0/documentation/PhysXGuide/Index.html [April 5, 2021].

PARKKULAINEN, E., 2019. *Real-time Physics Simulation*, Metropolia University of Applied Sciences.

ROBERT MCNEEL & ASSOCIATES, 2010a-last update, Curve Editor. Available at: https://bongo.rhino3d.com/page/curve-editor [April 30, 2021].

ROBERT MCNEEL & ASSOCIATES, 2010b-last update, Keyframes & Keyframe Editor. Available at: https://bongo.rhino3d.com/page/improved-keyframe-editor [April 30, 2021].

ROBERT MCNEEL & ASSOCIATES, 2010c-last update, Timeline. Available at: https://bongo.rhino3d.com/page/timeline-1 [April 27, 2021].

ROBERT MCNEEL & ASSOCIATES, 2010d-last update, Tweening. Available at: https://bongo.rhino3d.com/page/tweening-1 [April 30, 2021].

ROGERS, D.F., 2011. *An Introduction to NURBS: With Historical Perspective.* Academic Press.

RYOO, J. and KIM, E., 2015. *Big Data E-Book.* https://sites.psu.edu/bigdataebook/: .

SHIRLEY, P. and MARSCHNER, S., 2009. *Fundamentals of Computer Graphics.* 3 edn. CRC Press.

SHOEMAKE, K., 1991. Quaternions and 4 × 4 Matrices. In: JAMES ARVO, ed, *Graphics Gems II.* Academic Press, pp. 351-354.

THOMAS, S.W., 1991. Decomposing a Matrix into Simple Transformations. In: JAMES ARVO, ed, *Graphics Gems II.* Academic Press, pp. 320-323.

VAN DEN BERGEN, G. and GREGORIUS, D., 2010. *Game Physics Pearls.* CRC Press.