

Intuitive Rocket Science, the Game, the Paper

CIS 499 SENIOR PROJECT DESIGN DOCUMENT

Mark Henderson, Ben Kantor
Advisor: Stephen Lane
University of Pennsylvania

PROJECT ABSTRACT

Platforming has been a cornerstone of the video game industry since the early 1980's, and has existed in many different forms, from the classic side scrolling Mario Bros to the Free Running simulation Mirror's Edge. Platforming as a gameplay mechanic has been included in many genres, and has been adapted and changed for each genre.

One of these adaptations, that of using explosions to launch the player long distances (rocket jumping), has received very little professional attention, and is generally only accessible to people playing online multiplayer first person shooters like Team Fortress 2 and Unreal Tournament. Rocket jumping is usually a quick way to trade a player's health for an advantageous position behind enemy lines, but in every game it is well supported a group of players begin to create specific maps designed around this mechanic.

This project will, for the first time, create a platforming game based around rocket jumping. To make it more interesting and challenging, the game will be set in a stylized space environment similar to the setting of Mario Galaxy, allowing level designers to employ dynamic gravity and small planetoids.

The project's dedication to platforming over combat will allow for several other improvements for players. Aside from optimized controls and less inadvertent suiciding, players of Intuitive Rocket Science, the Game will enjoy alternatives to static maps with challenge after challenge. A survival mode will have players fleeing from imminent death over an endless field of procedurally generated planetoids, and if time permits a co-op mode will allow two players to complete jumps impossible to accomplish alone.

Project blog:

<http://intuitiverocketscience.blogspot.com/>

1. INTRODUCTION

A video game where game play elements of Mario Galaxy (jumping from small planetoid to small planetoid, each with their own gravity well) meets the rocket jumping popular in Quake, Unreal Tournament, and Team Fortress. This is the first game we are aware of to focus specifically on rocket jumping, instead of including it accidentally/incidentally.

1.1. Significance of Problem or Production/Development Need

Rocket jumping has been mostly ignored by professional game developers. While games like Team Fortress 2 has included it to appease to a base of players that took advantage of its less intentional existence in Team Fortress Classic, the controls are very awkward and the fans are left to create challenging maps on their own. The combination of keys that need to be used in rocket jumping (L-Ctrl, A, W, S, D, and Space) are awkward to use, and can cause cramps in the left hand. Since rocket jumping is usually incidental to the game developers, it is usually very difficult to do well.

These effects have combined to restrict rocket jumping to interested players of a few online FPS's willing to spend hours on end at a single challenge trying to press a sequence of keys with precise timing and in painful (for the hand) combinations.

Dynamic gravity platformers like Mario Galaxy are also still very rare, and allow for several new types of challenges. Very little exploration into this has been done, and creating a robust system in which fans can experiment with and create their own maps will allow many interested game players to experiment on their own.

1.2. Technology

C++
Gamebryo engine
PhysX
Maya

1.3. Design Goals

The project has two main goals:

Create a platforming game centered on rocket jumping that is accessible and playable by a larger audience.

Experiment with dynamic gravity platforming in new and interesting ways.

A secondary goal is to create a procedural modeling system to automatically create small planetoids with diverse geography.

1.3.1 Target Audience.

Typically, the target audience for rocket jumping is a small subset of online multiplayer first person shooter players. This game would target a much broader group of people, appealing to players who have never tried (or never been able to succeed at) traditional rocket jumping as well as experienced players who are looking for greater challenges.

Players of all experience levels, from those who have never heard of rocket jumping to those who have spent dozens of hours on the hardest rocket jumping challenges should all be able to install this and start having fun immediately.

1.3.2 User goals and objectives

Users should approach this with the intent to have fun. Possible bonuses include gaining an intuitive understanding of how gravity works in orbital systems (though much faster and denser than in real life), finally learning the basics of rocket jumping, and remaking the maps of other systems in this game to finally have a chance of beating them.

1.3.3 Technical Features

The game includes several technical features that the users might not fully notice, but will heavily effect game play.

Dynamic Gravity: This game will calculate gravity very similarly to real gravity calculations used for space travel, though very small planetoids will be generating gravity equal to earths, which will fall off only a few feet from their surface. Characters and objects will both be fully influenced by all the nearby masses, and it should be perfectly possible for a character to orbit a planetoid several times. Volumetric areas with preset gravity and direction immune to the effects of nearby masses will be available to game designers.

PhysX: for all the other physical aspects of the game, from collision detection to rag dolls, PhysX will be used. Explosions or other particles will be included if time permits.

Procedural Planetoid Generation: A tool to randomly create small planetoids with varying terrain will be available, likely based off current fractal height field mesh algorithms that can be applied to spheres.

1.3.3 Design Features

Several new design features intended to simplify and enhance rocket jumping will also be included.

Tutorial Mode: possibly a story mode, a set of pre-made levels will be included in the game, and will introduce new players to just what can be achieved with explosion based jumping. This is something that has never been generally available, as current rocket jumping maps have all been created by rocket jumpers looking for more of a challenge.

Survival Mode: An asteroid field stretches out before the player, and the entire thing is proceeding merrily into a star. How long can the player stave off imminent demise by jumping further away? Rather like swimming up a water fall, this mode will consist of an endless stream of procedural planetoids, all reasonably but randomly placed, which the player must use as stepping stones to stay away from death.

Optimized controls: As a dedicated platformer, this game can move more important controls, like crouch, to locations less likely to cramp a persons hand.

Custom propulsion gun: This system will have a gun designed to mimic the grenade jumps from TFC, stickybomb jumps from TF2, and rocket jumps similar to other games.

Graphics: The graphics for this system won't match up to the standards of a PS3 game, but should roughly match the quality of Blizzard's World of Warcraft; fully animated 3d characters in a colorful textured world with some basic particle effects.

2. Prior Work

Rocket Jumping:

Below is a short overview of the history of rocket jumping, taken from several online sources cited at the end of this paper.

Rocket jumping first appeared in Doom (1993), and only worked horizontally. A secret level exit was created by the developers, and though it can be reached by other means it is intended to be reached by rocket jumping.

Bungie Software's Marathon (1994) was the first game to allow players to launch themselves vertically as well as horizontally, but this ability was not used in gameplay.

Quake (1996) and its successors also included rocket jumping, though only as an accessory to combat gameplay.

Half-life's (1997) multiplayer mode allowed a limited form of rocket jumping, as did Unreal (1998) and Unreal Tournament (1999). However, it wasn't until Team Fortress Classic (TFC) released in 1999 that rocket jumping became a viable and platforming mechanic.

TFC and its successor, Team Fortress 2 (TF2)(2007) both allow players to create maps and share them with other players. Most created maps are combat oriented, but shortly after TFC's release several maps were created where players from opposing teams couldn't even reach each other.

The point of these maps was to race to the capture point at the end of the level, with the races often taking hours as players learned the fine art of using explosives to give themselves momentum. Combined with the limited air control (using normal movement keys to slightly change the player's direction while flying), very complicated challenges were soon developed. These became fairly popular over time, which lead to rocket jumping being included in TF2, where it continues to be developed and practiced.

Dynamic Gravity:

At this time, it seems that the only platforming game to have dynamic gravity is Mario Galaxy, which was the inspiration for this project.

In the greater field of science, a large amount of research has gone into precisely predicting gravity in space, and determining its effects on travelling space craft. The real world physics are well understood outside of black holes and faster than light conditions.

Procedural Planetoid Creation:

A great deal of work has gone into procedural terrain generation, often using height fields and fractal systems.

One major problem when wrapping height fields onto spheres is triangulation of the mesh at the poles. These artifacts are usually present when dealing with spheres of larger size and can go unnoticed, as in Google Earth's globe. The popular POV-Ray raytracer software approaches height fields by using Isosurfaces: a function that defines the terrain in the form of a three-dimensional topography map. By selecting certain threshold values and applying randomization, a wide variety of procedural maps can be created.

3. PROJECT DEVELOPMENT APPROACH

3.1. Algorithm Details

<will write more here when we know more, but there isn't much algorithm to the project. Will likely include the main game loop and the procedural planetoid generator.>

3.2. Tools used

3.2.1 Hardware

PC, graphics card that can run PhysX
motion capture system for animation

3.2.2 Software

Gamebryo
Maya
Visual Studio

4. WORK PLAN

4.1.1. Project Milestone Report (Alpha Version)

Split the game into two levels, the lower of these being a fully functioning game that only displays cubes and spheres, and the higher level being everything needed to go from the first level to a full game with avatars, animation, effects, etc. Mark Henderson will work mainly on the first level, while Ben Kantor will mostly work on the second level. Of course, there will be some overlap and sharing of tasks.

Mark:

Implement game engine, PhysX for objects
Create gravity simulation
Create controls / rocket launcher

Ben:

Create models, animations
Implement animation manager
Implement lighting, texturing, etc
Implement camera

Shared:

Thorough literature review
Procedural planetoid creator – basic implementation working

4.1.2. Project Final Deliverables

Split the game into two levels, the lower of these being a fully functioning game that only displays cubes and spheres, and the higher level being everything needed to go from the first level to a full game with avatars,

animation, effects, etc. Mark Henderson will work mainly on the first level, while Ben Kantor will mostly work on the second level. Of course, there will be some overlap and sharing of tasks.

Mark:

Implement game engine, PhysX

Create gravity simulation

Create controls / rocket launcher

Debug everything, iterate at least once.

Create 'survival mode'

(Maybe) PhysX for interesting explosions, other special effects

Ben:

Create models, animations

Implement animation manager

Implement lighting, texturing, etc

Implement camera

Debug everything

Level design – create basic tutorial/story mode to teach basic and advanced techniques

Shared:

Procedural planetoid creator

4.1.3 Project timeline.

Mark:

learn Gamebryo / physX, get (very basic) system running

basic system to include walking around, jumping, character/sphere collision

3 weeks

create robust and functional gravity system, rocket launcher system

1.5 week

create control schemes, main game loop

1.5 week

title screen, loading system, options, etc

1.5 week

-alpha-

debug/iterate

rest of time, will start alpha testing here

create survival mode

1.5 week

(if time permits, physX explosions should take a week or two)

Ben:

learn Gamebryo basics and functionality
model, texture, rig main character for use

3 weeks

implement animation manager and combine with control schemes

3 weeks

implement lighting and camera

1.5 weeks

-alpha-

debugging, etc.

tutorial and sample level design

1.5 weeks

time remaining: procedural topography

4.1.4 Gant Chart (attached)

5. RESULTS

Work Environment Setup:

- 1) Install DirectX 10 November 2008 version
- 2) Install NVIDIA PhysX System Software version (?)
- 3) Install NVIDIA PhysX SDK version (?) for Windows
- 4) Install Nima PhysX Maya 2008 plug-in
- 5) Install full Emergent Gamebryo 2.5 Evaluation version with Maya 2008 plug-in
- 6) Created project inside the Gamebryo Demo folders
- 7) In Visual Studio 2005, add the following Include files:
..\..\..\..\..\sdk\Win32\Include
- 8) In Visual Studio 2005, add the following Lib files:
..\..\..\..\..\sdk\Win32\Lib\VC80\DebugLib
\$(PHYSXINSTALLPATH)\SDKs\lib\win32\
..\..\..\..\..\sdk\Win32\Lib\VC80\DebugDLL
- 9) in Visual Studio 2005's tool options, add include files:

C:\Program Files\Microsoft DirectX SDK (November 2008)\Include
C:\Program Files\AGEIA Technologies\SDK\v2.8.0\SDKs\Core\include
C:\Program Files\AGEIA Technologies\SDK\v2.8.0\SDKs\Foundation\include
C:\Program Files\AGEIA Technologies\SDK\v2.8.0\SDKs\Cooking\include
C:\Program Files\AGEIA Technologies\SDK\v2.8.0\SDKs\Physics\include
C:\Program Files\AGEIA Technologies\SDK\v2.8.0\SDKs\NxExtensions\include
C:\Program Files\AGEIA Technologies\SDK\v2.8.0\SDKs\NxCharacter\include
C:\Program Files\AGEIA Technologies\SDK\v2.8.0\SDKs\PhysXLoader\include
and the lib files:
C:\Program Files\Microsoft DirectX SDK (November 2008)\Lib\x86
C:\Program Files\AGEIA Technologies\SDK\v2.8.0\SDKs\lib\win32

Work Breakdown:

Mark:

Programming:

- Created application framework
- Camera:
 - Ground based yaw and pitch camera with locked up axis
 - Flying yaw and pitch camera with no locked y axis
- Gravity:
 - Point source equivalent to real gravity
 - Constant gravity volumes to allow for areas that perform equivalent to normal fps
- Projectile that fires, applies forces to character and other projectiles

User Controls:

- Created initial settings
- Had several people play test the game, adjusted controls based on input [Mark/Ben]

Problems:

- Gamebryo documentation does not seem to be designed to be useful to people who have not paid \$4000 for the introductory class.
- Gamebryo would not read mouse input unless the left or middle mouse button was depressed.
- Ran out of time, so I couldn't implement a lot of options that would have taken relatively little coding, i.e. sticky grenades that explode on command instead of impact, level loading, etc.

Ben:

Test Level:

- modeled, textured
- applied Nima PhysX rigid bodies and gravity constants, which Mark then dramatically decreased to increase the difficulty

Main Character:

- modeled, rigged, animated in Maya
- created animation layers and sequences using Gamebryo plug-in
- set blending and sequence group weights and priorities in Animation Tool

Grenade Asset:

- same processes as main character, but also used Gamebryo particle system effects for fire and smoke explosion

Animation Class:

- centered on NiActorManager to import KFM files and add to scene graph
- after importing, character position is updated based on the PhysX ball, rotation based on camera
- scene graph root and ActorManager are given time variables to accumulate in the main Update() method and display animations
- keyboard and mouse presses are interpreted to know when to call, change, or cancel character animation sequences

Problems:

- Maya-exported KFM files fail to load in Visual Studio (including packaged samples), but Animation Class is easily tailored to any similar animated model by swapping out the [Char Name]_Anim.H file which uses an Enum listing of sequences made using the Animation Tool. It currently works with any Max-exported models, ex) the LM sample used in our demo. I've submitted a thread on the official Gamebryo forums to hopefully gain some insight. Currently downloading a trial of Max to try it that way.
- currently trying to implement sound, but requires additional commercial SDK package so researching workaround

6. WORK LOG

Sunday, May 10, 2009

After Mark and I finished up compiling our files and documents, we took yet another look at the KFM loading error. Again, no luck with Maya-exported files, so we decided to see if their only Maya sample was implemented in any demos. No. It is not used once.


So in a final act of desperation, I'm downloading a LEGAL free trial version of 3ds Max. Then I'll just install the Gamebryo plug-in and try to export it from there. Wish me luck!

Posted by Ben at [12:56 PM 0 comments](#) 

Saturday, May 9, 2009

Not having much luck with this. I went back and started to comb through the Documentation and 5 years of forum posts to try and figure out why functioning Maya KFM files refuse to load through the program. After testing some workarounds unsuccessfully, I left a question that will hopefully be answered sometime in the near future. In the meantime, I spent a few hours constructing a sound class, since Dr. Badler wanted to hear some explosions. After trying to

resolve some linking errors, I found this in the documentation: "Customers wishing to use this implementation must separately purchase a license for the Miles Sound System directly from RAD Game Tools, Inc." The MetalWars demo uses this, but of course, it won't compile on the system so I can't take the license from it. The documentation suggests that there are demos that use a different system, but I will have to look for them later. The issue seems to be that the NiAudio core library isn't included in this version.

Posted by Ben at [3:11 PM](#) [0 comments](#) 

Tuesday, May 5, 2009

Well, long haul is complete, presentation is done, and yet there's still so much more I want to add.

Here's a video of where the game was at for the project presentation on May 1st:

It's working much better than it has in the past, as I've worked out almost all the bugs. It doesn't have many features I'd like to add, but perhaps I'll have time this summer to expand on it.

Hopefully Ben will get the animation exporter to work correctly, so we can get rid of the gamebryo sample character featured in this video. That would also let us add the correct model and animation for the rockets.

Posted by Mark at [8:50 AM](#) [0 comments](#) 

Sunday, April 19, 2009

<http://www.youtube.com/watch?v=VeqrChzSIE>

new video up... bit more complex video with some interesting slingshot action :-)

need to fix the camera still, and the projectile code results in a wierd bug.

Posted by Mark at [5:24 PM](#) [1 comments](#) 

Friday, April 10, 2009


[camera and collision detection](#)

<http://www.youtube.com/watch?v=Gg5NZaLH4rE>

new video up on youtube which seems to work right.

I've got the collision detection working correctly now, so the character knows when it is flying or on the ground. The camera now has different controls based on flying/grounded, which are fairly intuitive to control. The transition between states needs work.

Hopefully, the next video I show will have a decent environment and (maybe) a character instead of a ball.

Posted by Mark at [7:39 PM 0 comments](#) 

Saturday, April 4, 2009

[Progress with a video](#)


Worked on the camera system, controls, and gravity a bit today.

I still need to automate the switch from flying around to walking around better, right now I press a key to change mode, and cause an instant camera change. Next step is getting collision information out of PhysX to automate the switch, and implementing a smooth camera transition function.

I still haven't tracked down the cause of the jittering frame. I still think somewhere in the code I'm updating something twice per frame update, but I can't find any places where that would happen. I've uploaded a [video](#) of where I'm at to youtube, and the bug is visible there, though it isn't as bad as normal. It seems that the bug gets better when the frame rate goes from 120ish to 60 (Fraps limits it for the video capture), since the bug is much more apparent when I'm not capturing.

I need to get Ben to make up an interesting level soon, I think I'm almost ready for some basic play testing.

-Mark

Posted by Mark at [7:18 PM 2 comments](#) 

Saturday, March 28, 2009

[complex scenes](#)

Made the system able to load more complex scenes, though the gravities still need to be hard coded. Worked on the camera and other controls too.

Posted by Mark at [5:58 PM 0 comments](#)

Tuesday, March 24, 2009

[sorta working!](#)

Sorry for the hiatus, Been working on other projects for a couple of weeks.

Physx has been integrated and working, and I managed to walk around and do a bit of jumping around a sphere, so my gravity system seems to work. need to fine tune the variables for that a bit, and make sure the gravity force doesn't exceed the character's ability to walk and jump.

Next step are implementing a bunch of the control niceties needed for a game, like a camera that stays pointing intelligently, and jumping that actually goes away from the standing surface instead of +y

Posted by Mark at [1:23 PM 3 comments](#) 

Tuesday, February 24, 2009

Bits and Pieces

I've been working on creating all of the art assets for the first part of the project, so there hasn't been much room for programming yet. Mark and I tested out exporting proxy models for the main character, and even then it took a while and caused performance jumps due to the amount of geometry. We're going to try and avoid spheres to cut down on faces, but smoothed cubes make for rather lumpy planetoids. Lighting is also working, but we have to use specific Gamebryo assets or else they get ignored when exported.


Currently I'm working on finishing the main character model so that I can create a KeyFrame file for it to start programming animation sets. Gamebryo doesn't allow you to edit .nif files much after you've exported, so it's very important to put in the extra effort in this stage to avoid problems later. The Animation Editor is fairly intuitive so I didn't spend too much time messing with demos. More on that once the .kf file is ready.

Posted by Ben at [10:38 AM](#) [0 comments](#) 

Monday, February 23, 2009

Camera is working great, I can export models from Maya to my app, gravity is programmed (but not tested), and I need to decide between PhysX and Gamebryo collisions.

Gamebryo will probably be easier to control for my own stuff, but PhysX will be better for objects that need to bounce around. I wonder how hard it would be to do some of each, and still have them interact? Probably harder than I want to deal with, but it's something to look into.

Posted by Mark at [6:16 PM](#) [1 comments](#) 

Sunday, February 15, 2009

Camera

Spent a few hours today figuring out camera and mouse/keyboard ui in Gamebryo. I have most of a basic system for flying around working, just need to debug it then adapt it for running around spheres and having alternate modes to distinguish running and jumping.

Posted by Mark at [9:09 PM](#) [0 comments](#) 

Monday, February 9, 2009

Alternative project idea

Probably better suited to a pure CS major than me, but something that really needs to be made:

A better way to manage linking multiple libraries together, so people like me don't have to spend hours trying to figure out all the different directories that need to be added to the project settings.

Maybe something that attempts to compile it, reads through the errors, and searches your computer / the internet to figure out what needs to be linked and what needs to be installed?

The current system has been a struggle for me every time I've had to install new SDK's onto a machine, and is almost as frustrating as debugging, only without that rewarding feeling of a working system you get when the bug is fixed.

Anyways, I've finally gotten Gamebryo to compile, though PhysX is still complaining about a bunch of classes not being defined (classes like NxClothMesh, which apparently don't appear anywhere in the SDK files). I'm just going to skip that for now and get some basic stuff running in Gamebryo, get back to PhysX when it matters and there are people nearby who have gotten it to work before.

Posted by Mark at [2:59 PM](#) [2 comments](#) 

Monday, January 26, 2009

[Setting up](#)

Finally have the Gamebryo demos running, and I've managed to launch a blank window with my own code.

Now lets see about installing physX...

Posted by Mark at [1:23 PM](#) [4 comments](#) 

Wednesday, January 21, 2009

[Project Proposal statment](#)

Might as well post this:

Intuitive Rocket Science, The Game

Procedural Planetoids and Non Static Gravity Physics

Objective

A video game where game play elements of Mario Galaxy (jumping from small planetoid to small planetoid, each with their own gravity well) meets the rocket jumping popular in Quake, Unreal Tournament, and Team Fortress. This is the first game we are aware of to focus specifically on rocket jumping, instead of including it accidently/incidentally.

Results Expected

A fully working game engine to include the following:

- procedurally generated planetoids/asteroids
- realistic* physics for jumping, with the option of blowing yourself up to propel you further (fire a rocket at an object near you, ride the explosion)
- realistic* physics for extra objects in the scene (barrels, boxes, other players, etc)
- first person or intelligent over-the-shoulder camera setups (players choice)
- Fully modeled and animated character/s with blending between animations and (possibly) rag doll behavior.
- bazooka/grenades that can be used to propel yourself mid jump
 - this would require explosions, of course, and it would be really nice if we could implement an actual explosion simulation instead of using a sprite of some sort. We will look into using the built in PhysX for this.

- (possibly) multiplayer over LAN, internet (we have no clue how networking works, so we're not sure about that.

- If we can't do multiplayer, there will be some other form of challenge, possibly outrunning a bunch of aliens or something.

* the gravity being generated by the asteroids would be far more than an object of that size could possibly generate, and would fall off far faster, but I aside from that we intend the physics and collisions to be accurate.

Manner of presentation of results

Blog to be updated at least weekly

Working game with sample levels and videos of gameplay

References

Will be using the Gamebryo engine, which includes PhysX, and associated documentation

(need to research SIGGRAPH papers about implementing height field meshes on spheroids)

Posted by Mark at [3:23 PM 0 comments](#)

7. REFERENCES

All your references list here.

http://www.bookrags.com/wiki/Rocket_jumping

http://wiki.quakeworld.nu/Rocket_Jump

<http://www.emergent.net/en/Support/Gamebryo-Textbook/>

<http://www.econym.demon.co.uk/isotut/index.htm>