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 then 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 player
s 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)

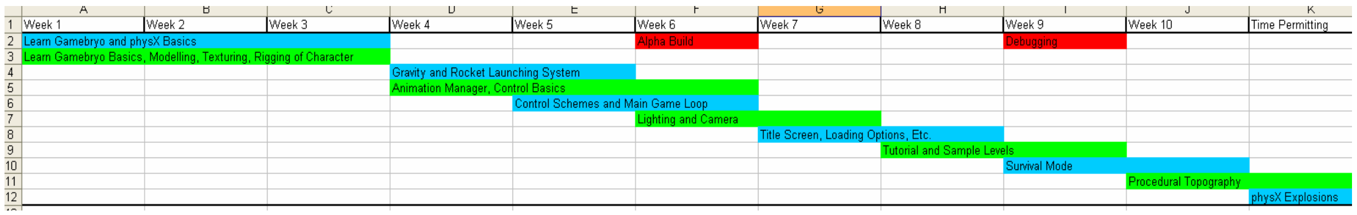
Ben:
learn Gamebryo basics and functionality
model, texture, rig main character for use
3 weeks
implement animation manager and combine with control schemes
3 weeks
implement lighting and camera
1.5 weeks

-alpha-

debugging, etc.
tutorial and sample level design
1.5 weeks

time remaining: procedural topography

4.1.4 Gant Chart



	A	B	C	D	E	F	G	H	I	J	K
1	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Time Permitting
2	Learn Gamebryo and physX Basics					Alpha Build			Debugging		
3	Learn Gamebryo Basics, Modelling, Texturing, Rigging of Character										
4				Gravity and Rocket Launching System							
5				Animation Manager, Control Basics							
6					Control Schemes and Main Game Loop						
7						Lighting and Camera					
8							Title Screen, Loading Options, Etc.				
9								Tutorial and Sample Levels			
10									Survival Mode		
11										Procedural Topography	
12											physX Explosions

5. 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