Lights Out

Non-Euclidean Game Engine and Video Game

Tasman Grinnell, Ben Johnson, Josh Deaton, Cory Roth, Zach Rapoza, Spencer Thiele, Lincoln Kness

Group: sdmay25-37

Faculty Advisor: Dr. Joseph Zambreno **Client:** Josh Deaton

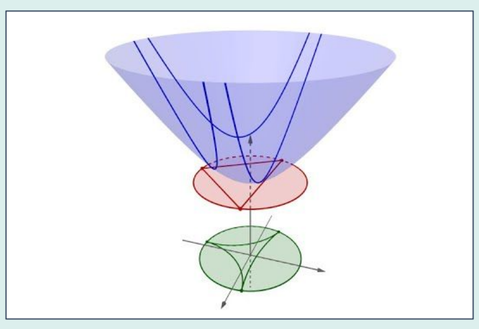
Introduction

- Existing game engines **DO NOT** support Non-Euclidean geometric models
- "Non-Euclidean" in industry commonly uses portals and disappearing rooms instead of a **Non-Euclidean Space**

We created a custom game engine and video game using custom Non-Euclidean hyperbolic shaders.

What is Hyperbolic Geometry?

A world that exists on a hyperboloid, but projected onto a 2-dimensional disk.



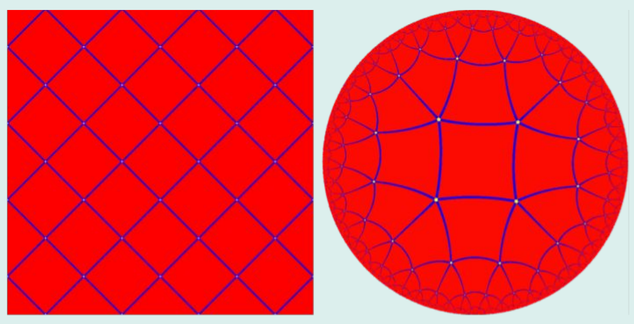


Figure 1 (top left): Hyperbolic space projection on a 2D disk

Figure 2 (top right): Euclidean vs hyperbolic square tiling

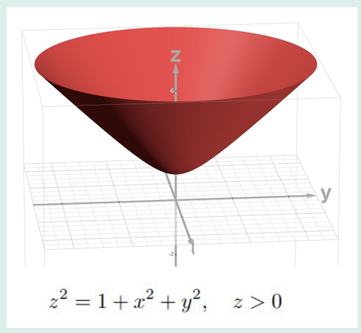


Figure 3 (bottom right): Two-sheet hyperboloid

Context

Users

- All types of gamers:
  - Casual and hardcore
- Game developers
- Researchers

Use Cases:

- Niche game development
- Non-Euclidean shader modeling
- Simulations

Requirements

Engine:

- Smooth gameplay on low-end hardware
- Intuitive API for game developers
- Support game design goals

Game Design:

- Emphasize Non-Euclidean shaders
- Enjoyable and interesting game mechanics
- Engine compatibility

Game Engine

Design Approach

- Entity-Component-System paradigm – operates on **parts** rather than a **whole** object
- Custom system scheduler for optimized updates
- Modularized and scalable code
- "Don't build functionality from scratch!"

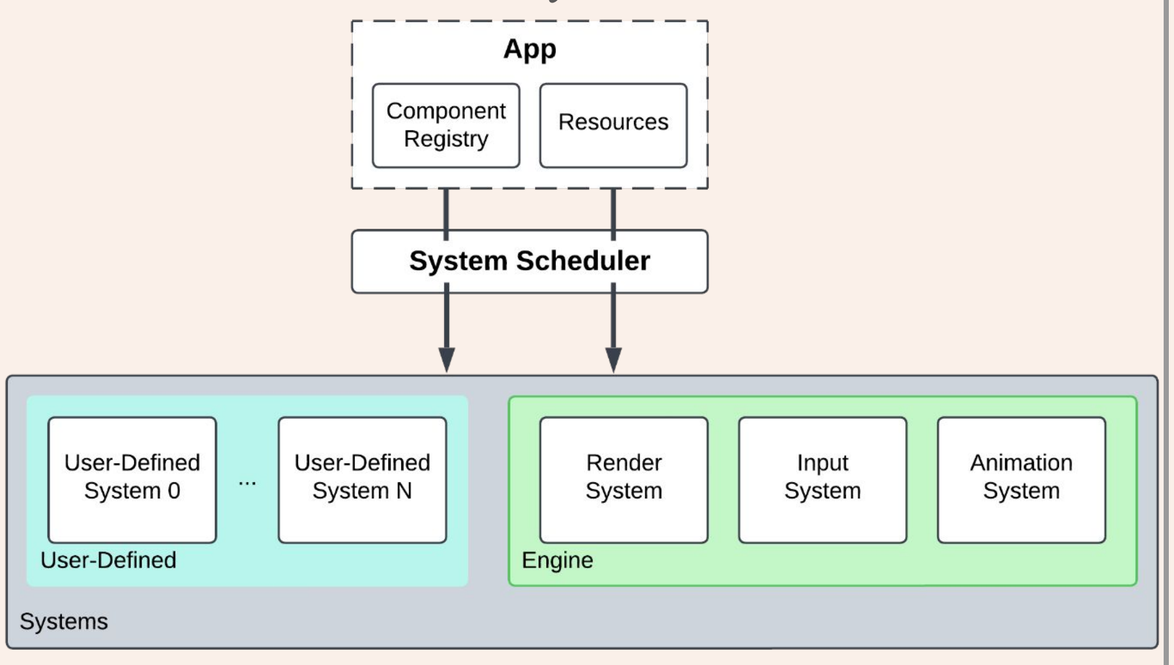


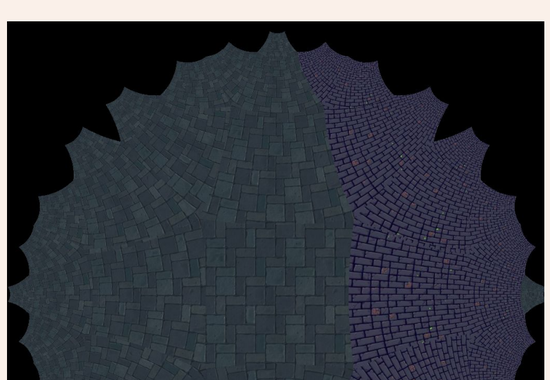
Figure 4: High-Level overview of game engine operation

Technical Details

- Written in C++ with OpenGL GLSL shaders
- CMake and Ninja – building/compiling tools
- External libraries – "don't reinvent the wheel!"
- Custom Non-Euclidean shaders

Testing

- Manual testing and unit tests – functionality focus
- Integrating tilemaps with shaders
  - Identify performance issues
  - Visually confirm sprite warping



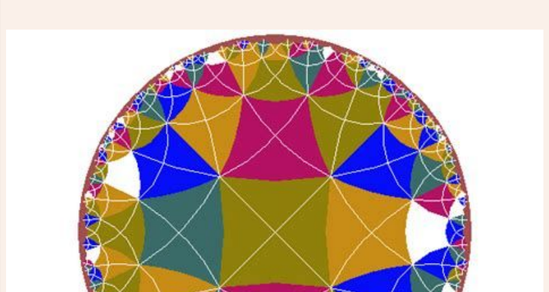


Figure 5: Tile maps showcasing hyperbolic shaders and warping on a sprite with solid color tiles

Game Design

Toolchain

- Figma/Docs → Brainstorming, scheduling (maybe toss)
- Unity (C#) → Prototyping and scene management
  - Testing while developing
- Git/Github → Version control
- Itch.io → Build the demo for playtesting
- Non-euclidean engine → port to custom engine

State of the Game

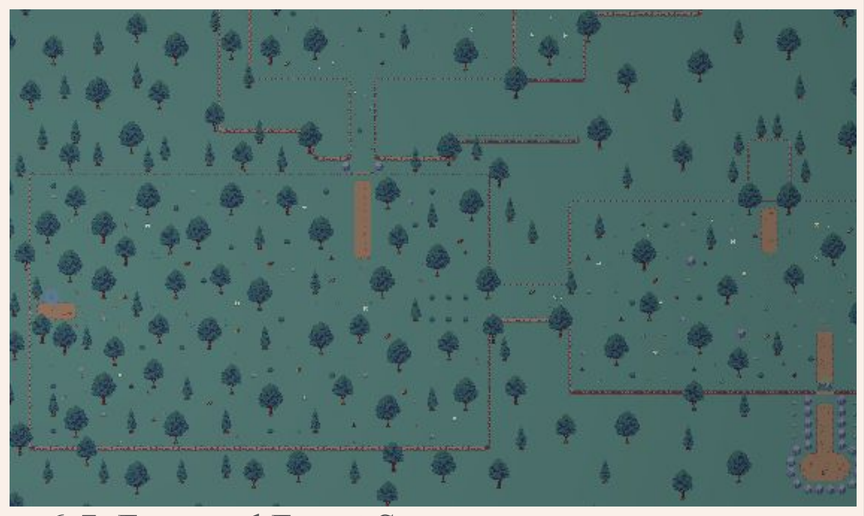


Figure 6-7: Farm and Forest Scenes

- Goal: Find your missing grandfather
- Primary mechanics – Planting, trading, and exploring

Play Testing - Itch.io Build

Issues

- Enemies were easily avoidable
- Interaction were unclear
- Lots of accidental inputs

Potential Solution

- Randomize the enemy spawns
- Add visual indicators
- Separate input buttons

Engine-Game Integration

- Early porting process
- Modified the forest scene to be more mazelike

Current issue: converting tile maps from the Unity prototype to custom tile map solution.

Existing conversions are time intensive (manual creation) or memory intensive (python scripting).

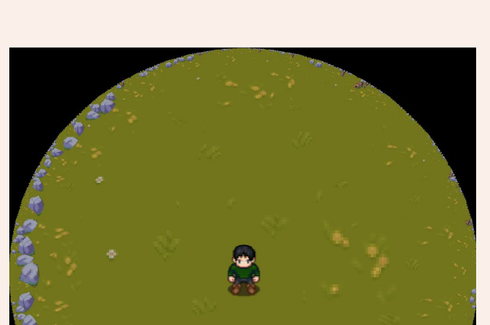


Figure 8: Integrated Forest/Maze