

Documentation Do(d)geBall

Table of Contents

Documentation Do(d)geBall

- Table of Contents
- Gameplay
- Controls
 - Player Controls
 - Miscellaneous
- Implementation
 - GameObjects and Components
 - Models and Meshes
 - Shading
 - Views
- Basic Effects & Gameplay
- Optional Effects & Gameplay
 - References
- Used Libraries

Gameplay

2 players play as doges in a dog park and throw frisbees at each other. When a frisbee hits a doge it disintegrates loses a life and respawns.

Controls

The game is played locally on one keyboard (with Numpad) and two mice. The following keys' names are derived from a German keyboard layout.

Player Controls

Key (Player 1)	Key (Player 2)	Action
W	Pos1	Move Forward
A	Entf	Move Left
S	Ende	Move Backwards
D	Bild ↓	Move Right
Left Shift	Right Shift	Sprint
Space	Numpad 0	Jump
Mouse 1 Left Mouse Button	Mouse 2 Left Mouse Button	Throw Frisbee

Miscellaneous

Key	Action
Esc	Exit Game
F1	Toggle Wireframe
F2	Toggle Backface Culling

Implementation

GameObjects and Components

The core components of the engine are GameObjects. They are hierarchically structured, and represent the objects in the game such as the players, the map, the projectiles and everything else. Attached to them are various Components which dictate their behaviour.

Models and Meshes

Models consist of multiple meshes with their respective transformations and materials as well as the shader which should be used for rendering. Meshes are modelled with Blender, exported using the Collada file format and loaded using the Open Asset Import Library (Assimp).

Shading

The game uses physically-based shading based on the Cook-Torrance model using the following mathematical terms:

- GGX Micro-facet distribution
- Geometric attenuation term
- Schlick's approximation of the Fresnel term

Materials can have a color, roughness, metalness and an ambient illumination.

Views

Multiple views are achieved by changing the viewport and using multiple cameras (with different view matrices) for each view.

Basic Effects & Gameplay

- 3D Geometry
- Playable
- Advanced Gameplay
- 60 FPS Framerate Independence
- Win/Lose Condition
- Intuitive Controls
- Intuitive Camera
- Illumination Model
- Textures (visible in skybox)
- Moving Objects
- Documentation
- Adjustable Parameters

Optional Effects & Gameplay

Feature	Description (Usage)
Collision Detection (Basic Physics)	Basic physics for the player characters (no walking through walls or falling through the floor), jumping
Advanced Physics	Physics for projectiles - drop off, bouncing and hit detection
Heads-Up Display	Display start and end screen during the game the player of a view and the current health of the players.
Lighting: PCF Shadow Mapping	Shadows add a more realistic lighting feel to the scene.
Animation: GPU Vertex Skinning	The characters movement is animated in blender and then exported in a collada file which is then read and used in the vertex shader to animate the characters in game.
Post Processing: Bloom/Glow	The projectiles have reflective surfaces which shine (and bloom) when at the correct angle to the light source.
Texturing: Env. Map	The Environment Map adds a skybox to the game to make it feel like the game lives in an actual world.

References

- <http://learnopengl.com> for effects.
- <https://www.youtube.com/watch?v=cieheqt7eqc> for animations

Used Libraries

Name	License	URL
Assimp	BSD	https://www.assimp.org/
GLEW	BSD	http://glew.sourceforge.net/
GLFW	zlib/libpng (BSD-like)	https://www.glfw.org/
OpenGL Mathematics (glm)	The Happy Bunny License (Modified MIT License)	https://github.com/g-truc/glm
NVIDIA PhysX SDK	BSD	https://github.com/NVIDIAGameWorks/PhysX
spdlog	MIT	https://github.com/gabime/spdlog
stb	MIT	https://github.com/nothings/stb
stduuid	MIT	https://github.com/mariusbancila/stduuid
INIReader	BSD	https://github.com/benhoyt/inih
FreeType	FTL (BSD-like)	https://www.freetype.org/