

# Submission 2 Documentation - The Chosen Frog

Reiser Simon

June 2019

The game currently only works with a NVIDIA GPU currently!!

## 1 Features and Implementation

Since I am developing on a Linux machine, thechosenfrog is a CMAKE project, that contains most dependencies as source, which are built together with the actual project source to ensure cross-platform compatibility.

The game is updated using a fixed time step. To achieve frame rate independence, the game is only updated until enough time is accumulated or updated multiple times if too much time has passed since last frame. The accumulator is capped to a maximum such that there is a maximum amount of updates that can be performed per frame, without this, slow machines would not be able to catch up with updates needed to perform. Therefore the game may slow down on slow machines.

### 1.1 Gameplay

- **First Playable:** The player has to run and jump on procedurally placed platforms to reach the red cloud.
- **3D Geometry:** The game uses a signpost model loaded from an .obj file. The loading of models is done using the assimp library. The core loading code can be found in *Game/src/assets/MeshList.cpp*.
- **Textures:** The game has support for diffuse textures and almost all meshes contain uv mapping information, but the game sparingly uses textures. The skybox and text are the only textured objects, which can be seen in the game.
- **Win/Loose Condition:** The game features a win and loose condition. When falling off the platforms, the player loses and the game can be restarted using F11. The player wins by reaching a red platform two times.

Players can challenge themselves by reaching the top as fast as possible or with the least amount of jumps. Both information is shown at the top platform and in the heads-up display.

- **Intuitive Controls:** The first version of the running and jumping have been implemented. The PhysX Kinematic Controller is used to perform position integration and collision detection, velocity and acceleration of the player is not managed by PhysX.

Responsible Files and Methods:

- *Game/src/components/CharControllerComponent(.h/.cpp)*
- *Game/src/EventManager.cpp*
- *Game/src/GameUpdater.cpp*

- **Intuitive Camera:** The movement of the camera is done like in most first person shooters. To achieve this, the ability to add objects/transforms as children of other objects/transforms (tree structure) has been implemented (see *Game/src/components/TransformComponent.cpp*)

The camera object is a child object of the player, thus always follows the player and only manages rotation around the (local) x axis.

Responsible Files and Methods:

- *Game/src/components/CameraControllerComponent(.h/.cpp)*
- *Game/src/GameUpdater.cpp*

- **Documentation:** You are currently reading the documentation of the game.

- **Adjustable Parameters:** The following parameters can be changed by setting the values in the settings file located at *assets/settings/settings.ini*:

- Window Width
- Window Height
- Fullscreen
- Refresh Rate
- Brightness

By default the game may not utilize the full width of the window, since the field of view may be too high in my opinion and the gameplay does not need a big horizontal view, since it's mostly vertical.

But there are also some viewport settings in the gameplay settings, which can be accessed by pressing **Ctrl + 4**. (Press **F8** to hide/show cursor), where the viewport behaviour can be changed.

- **Physics Engine:** The game currently uses PhysX to perform collision detection and position integration (movement) for the player controller.

- **Heads-Up Display:** In the bottom left corner, the time is shown as well as the amount of jumps performed.

## 1.2 Effects:

- **Vertex shader animation:** When the player charges up a jump (see Controls), the jump that would be performed by releasing the jump key is visualized by showing a thick, transparent dashed parabola. The dashes are moved over time along the jump parabola.

The dashes are actually not animated in the vertex shader, but in a geometry shader, since the dash geometry is calculated on the fly. The game only passes a vertex buffer with values ranging from 0 to 1 to the gpu to tell at which positions (in parameter space) of the parabola a dash should be displayed. The vertex shader simply forwards this value and the geometry shader changes the value over time calculates the world position of this point and generates the geometry for the dash.

Responsible Files and Methods:

- *Game/src/GameRendererer.cpp*
- *Game/assets/shaders/parabola(.vert/.frag/.geom)*

Also, the vertices of the clouds are animated, they grow and shrink a little bit over time using a sine function.

Responsible Files and Methods:

- *Game/assets/shaders/cloud.vert*

- **Contours via Edge Detection:** The game renders contours using both depth and normals (color is not used). Normal contours on clouds fade in when the player is close enough, since the outline would be too big for far away objects.

To achieve contours the scene is rendered with two render targets to a frame buffer object. The normal scene is rendered into the first render target and the view space normals into the second one. Then, the texture from the first render target is rendered onto a quad mapped directly on the screen and the contours are rendered on top afterwards using the texture from the second render target as input for the post processing shader.

The post processing shader uses a Roberts Cross Kernel with variable size to control the outline size.

Responsible Files and Methods:

- *Game/src/GameRendererer.cpp*
- *Game/assets/shaders/fbo(.vert/.frag)*
- *Game/assets/shaders/fboPost.frag*

## 2 Controls

### 2.1 Player Controls

- **W A S D:** Movement of the frog - W to move the frog forward, A to move the frog to the left, S to move backwards, D to move to the right. The frog can also be moved in the air up to a certain degree.
- **Space:** Jump - When pressing the space key the player charges up a jump, which is performed upon releasing the space key. The longer the space key has been hold, the farther and higher the jump will be.  
  
The looking direction also influences the jump, if the player looks up then the jump will be higher, but less far, if the player looks straight, the jump will be far but not high.  
  
Upon holding the space key, the jump, that will be performed, is visualized with a parable indicating how strong the jump will be and where the frog will land, assuming the player does not influence the jump by pressing the movement keys.
- **Mouse:** Camera Rotation - The camera will rotate through solely moving the mouse up and down or left and right.
- **Right Click:** Cancel Jump - Pressing the right mouse button while charging up a jump, cancels it.

### 2.2 Debug Controls

- **F11:** Restart game
- **F10:** Toggle Backface Culling
- **F9:** Toggle Wireframe Rendering (This does not really work currently, since you only see the quad where a Frame Buffer Object is rendered onto)
- **F8:** Hide/Show Cursor Toggle
- **Ctrl + 1:** Toggle debug logging window
- **Ctrl + 2:** Toggle debug component viewer window
- **Ctrl + 3:** Toggle debug asset viewer window
- **Ctrl + 4:** Toggle game and display settings window

## 3 Illumination

Currently, no lighting has been implemented. Support for diffuse textures has been implemented.

## 4 Used libraries

- assimp - Assimp is used for loading models (in our case from .obj files)
- PhysX - PhysX is used for collision detection and movement
- glad - glad is the OpenGL Function Loader used.
- glfw - GLFW is used for window management and providing time and input from the underlying system
- glm - We use the vector and matrix types and operations provided by glm.
- imgui - imgui is a light-weight immediate mode GUI library used for creating debug GUI.
- stb\_image - stb\_image is a small, header-only library we use for loading images from the disk.
- stb\_truetype - stb\_truetype is a small, header-only library we use to load true type fonts, generate bitmaps and load font information.
- soloud - SoLoud is used for loading and playing sound and music (currently unused)
- cute\_filewatch - cute\_filewatch is a small, header-only library for watching directories for file changes. This will be used for hot reloading assets while the game is running, but is currently not fully functional in this submission.
- RuntimeCompiledCpp - RuntimeCompiledCpp is a system, that makes it possible to recompile parts of the application while it is running. This is currently not functional.