Obstacles

Submission 2

Philipp Kafka (1027074), Thomas Lang (1025705)

June 25, 2013

1 Features

- skybox
- rotatable and zoomable camera (mouse controlled)
- controllable player character (keyboard controlled)
- physics
- objects to interact with (obstacles)
- textures
- illumination
- basic level file format and loader
- animations
- bloom
- motion blur
- soft, dynamic shadows

2 Controls

The camera can be rotated by moving the pointer while holding down the left mouse button. Camera zoom is controlled through the mouse wheel. The player can be moved using the A and D or the arrow keys. To jump, press space.

2.1 Debug Controls

- *F1* – Help
- *F2* – FPS Counter on/off (visible in window title bar)
- *F3* – Wire Frame on/off
- *F4* – Textur-Sampling-Quality: Nearest Neighbor/Bilinear
- *F5* – Mip Mapping-Quality: Off/Nearest Neighbor/Linear
- *F6* – Bullet Physics Wireframes
- *F7* – Stop Player Movement (Zero Velocity)
- *F8* – Viewfrustum-Culling on/off
- *F9* – Transparency on/off

3 Command Line Parameters

- `--fullscreen` to open game in fullscreen (default in release mode)
- `--windowed` to open game in a window (default in debug mode)
- `-w <width>` for specifying a width
- `-h <height>` for specifying a height
- `-v <value>` for controlling vertical synchronisation

Width/height parameters have both to be set in order to take effect. In windowed mode these refer to window dimensions. In fullscreen mode the nearest supported resolution matching them is selected. Without both these parameters the desktop resolution is used in fullscreen mode while in windowed mode the default window dimensions are.

4 Instructions

The goal of this game is to reach the finish line of every level as quickly as possible.

In order to reach the second level one has to reach the finish line without falling into the abyss. Space bar is key, some ramps are best to be avoided, be wary of those duckies.

If you fall down, you start at the beginning of the level. The time only stops at the

end of the game (when all levels are completed), so each fall costs precious time.

5 Implementation

The in-game world is represented by a scene graph containing all the objects that are present in the scene. Every object inherits from the SceneObject type and can contain other SceneObjects. The update and draw methods of a SceneObject are propagated to all the child objects.

Additionally, we are using a lot of helper objects like the model loader, level loader, shader manager and so on.

5.1 Player and Object physics

Collision detection and all movement in the game is controlled by the physics engine (Bullet). The player is a dynamic physical object that is controlled by forces (for the movement) and impulses (for jumping). The floor consists of several platforms that are separated by gaps over which the player has to jump some way or the other.

Every object that gets drawn on the screen contains at least one Geometry object which does the actual OpenGL work like filling buffers and making the draw calls. The geometry data for that is loaded from model files using the Assimp model loader.

5.2 Animations

Every transformable scene object can have one or more animations applied to it. The animations are just functions that are stored in a list inside the object and can simply be added by use of lambda expressions. Model transformations also apply to their individual meshes, therefore animations that transform models are hierarchical. An animation like that can be seen when the player moves over a duck. When in the air, the player's legs still rotate on their own but are also affected by the spinning of the whole character.

5.3 Levels

Levels are stored using a custom level file format and can be found in the levels/ directory. The format is documented in the level files themselves using comments. There is also a file named "levels.txt" which stores the level succession. At the start of the game and at the end of every level, the loadNextLevel method from the level loader gets called.

This method reads the next line from `levels.txt`, parses the corresponding level file and constructs a Level object accordingly.

The level file format describes the size of the level, the setting (day or night), the platform and gap sizes and all the obstacles including their positions (at the moment only cubes are possible).

5.4 Models

All Models but the Ducky model were created using Blender, they are all stored in the Wavefront object format.

5.5 Textures

We are also using "Ducky" from ECG with its textures. The obstacles have a custom texture that we made. The floor uses a free-to-use texture, the link to the source can be found in `textures/ground.txt`. For the UI we use only self-made textures (e. g. the help screen), even for the fonts (created using Luxi Mono font).

5.6 Illumination and Shading

Currently there are two level settings that can be configured in the level files: *day* and emphnight. All lighting is done using Phong shading.

In the day setting, a directional light source is used in addition to bright ambient lighting. In the night setting, there is a point light hovering over and slightly behind the player that illuminates the way. The light from it gets attenuated partly in a linear and partly in a quadratic fashion. In the game engine, the spot light is just another scene object that is transformed like every other object. Both day and night also include Blinn-Phong specular reflections. The UI uses transparency.

5.7 Effects

- Thomas:
  - *Bloom*
    The Bloom implementation follows the slides from the CG UE Repetitorium [1] and is only used in day levels. Threshold for the bright pass is pretty low (0.3), which has the side effect that the color intensities also increase. For

filtering, a 7x7 gaussian kernel is used (horizontal and vertical pass). In the blending pass, the resulting picture is calculated as $original + 3 \cdot filtered$.

  - *Motion Blur*
    The Motion Blur effect was implemented using the GPU Gems 3 Tutorial from Nvidia [2]. It is active in both day and night levels. The effect consists of two parts: Camera Motion Blur and Object Motion Blur. Camera Motion Blur only happens when the camera moves (but when the player moves, the camera follows it automatically). It is calculated using the depth buffer (texture) of the already drawn scene. For Object Motion Blur, fragment positions are calculated for the current and for the previous frame. The differences are stored as velocity values in a velocity texture that is then used in the motionBlur fragment shader.

- Philipp:

  - *Variance Shadow Mapping*
    Basic first shadow mapping implementation (targeted to become shadow mapping with PCF) done with the help of the official CGUE revision course with its slides [3]. Then changed into VSM implemntation (quite some changes) taking advice from [4] and [5]. In the depth pass where all geometries uses special shader (and skybox isn't drawn at all) the depth and the squared mean of a distribution of depths (derivatives along x and y of the view from the light source) are stored into an two-channel texture (therefore needing to make use of the color and not the depth attachment of framebuffer) which is blurred in another pass. In the render pass the fragment shaders then retrieve the two values (moments) from shadow map from the corresponding position and uses them using Chebyshev's inequality for calculating a probabilty of being shadowed which happens to be a good method for drawing a shadow with (pseudo-)penumbra.

6 External Dependencies

- *GLFW (window managing, OpenGL runtime)*
  Version: from repo after 2.7.8
  URL: `http://www.glfw.org/`

- *FlextGL (OpenGL extensions)*
  Version: dad6eb4fc3
  URL: `https://github.com/ginkgo/flextGL`

- *GLM (math)*
  Version: 0.9.4.4

URL: http://glm.g-truc.net/

- *Assimp (model loading)*
  Version: from repo after 3.0
  URL: http://assimp.sourceforge.net/
- *FreeImage (image loading)*
  Version: from repo after 3.15.4
  URL: http://freeimage.sourceforge.net/
- *Bullet (physics)*
  Version: 2.81
  URL: http://bulletphysics.org
- *zlib (for FreeImage and Assimp)*
  Version: 1.2.7
  URL: http://zlib.net/
- *getopt_long Windows port* (command line options parsing)
  Version: Oct 2012
  URL: http://www.codeproject.com/Articles/157001/Full-getopt-Port-for-Unicode-and

7 General Resources

- https://lva.cg.tuwien.ac.at/cgue/wiki/doku.php?id=students:introduction_talks_code
- https://lva.cg.tuwien.ac.at/cgue/wiki/doku.php?id=students:debugcontext
- http://arcsynthesis.org/gltut/
- https://sites.google.com/site/drunkdevsltd/tutorials/draw-a-sphere-using-opengl-

References

[1] CGUE Repetitorium: Bloom, https://lva.cg.tuwien.ac.at/cgue/wiki/lib/exe/fetch.php?media=students:cgue13_bloom.pdf

[2] GPU Gems 3, http://http.developer.nvidia.com/GPUGems3/gpugems3_ch27.html

[3] CGUE Repetitorium: Shadow Mapping, https://lva.cg.tuwien.ac.at/cgue/wiki/lib/exe/fetch.php?media=students:cgue13_shadowmapping.pdf

[4] Variance Shadow Maps, http://www.punkuser.net/vsm/

[5] Softshadow with GLUT, GLSL and VSM, http://fabiensanglard.net/shadowmappingVSM/