Hive – 2nd submission

Graphics

- Lighting model that supports ambient light, diffuse textures, specular lighting, specular maps, and normal mapping
- Multiple (up to 50) omni- and directional light sources rendered simultaneously
- Debugging shader that displays normals, tangents and bitangents using a geometry shader
- Framebuffer for multiple shader stages
- Dilation shader that paints the silhouette of a mesh
- Bone animations with vertex skinning calculated in the vertex shader
- Multiple animations influencing different bones, played independently
- 2D and 3D interface rendering
- Wireframe rendering
- Particle system with depth-sorting and transparency
- Frustum culling

Logical functions

- Texture loading from TGA or PNG files
- Mesh and bone animation loading from Collada files
- Mesh serialization for shorter loading times
- Scene graph with hierarchical transforms in the internal nodes and meshes and lights in the leaf nodes
- Sweep and prune algorithms for efficient collision detection in the entire scene
- Collision algorithms for different primitives (axis-aligned and oriented bounding boxes, frustums, cylinders, rays, triangles, planes etc.)
- Event system for mouse and keyboard and timed events
- NURBS curves

Game content

- Geometry for a three-leveled cone
- High polygon skeleton model to represent enemies and the player
- Models for sword, boots, armor, helmet, scrolls, crate, ladder, gate
- The player can move around and attack enemies with a sword
- Simple enemy AI
- UI for storing equipment and spell scrolls
- Spell that uses the particle system

Gameplay

The player can be moved around with WASD. When standing close to an item on the floor, it will flash. When the mouse is hovered over it, its silhouette will be painted. If you click on the item it will be placed in your inventory automatically. You can also drag it to a suitable slot.

Enemies form small battalions and will patrol between two points on the map until the player comes close enough for them to see him. Then they will try to surround and attack him from different sides. When they flash yellow they are about to attack, which means the player should either attack them (LMB) right away or evade. When they are hit they will be stunned for a short time. When an enemy is killed he will leave all his equipment behind for the player to loot.

The player can carry to hand weapon sets which can be switched be pressing 'C'. (Since there's only one kind of sword that doesn't really help you though.)

Along the way you will also encounter crates which will fall apart and spill their contents upon a hit.

At they end of a level you will find a ladder allowing you to descend into the next ring.

Demo Walkthrough

- Pick up the sword
- Walk to the right
- Kill the first five enemies and take the armor and helmet that one of them drops
- Destroy the crate and take boots
- Take on the next three armored skeletons
- Destroy the next crate and get a scroll for a pain spell
- Activate the pain spell by pressing '1' (or any key from '1'-'4' depending on where you put the spell)
- Kill the next battalions
- Go down the ladder
- Wait for the presentation to see the boss (if we finish it in time...)

Requirements

- Gamplay: The game logic was implemented in a World class which controls all the entities in the game. It makes sure to update and draw all entities as well as to resolve collisions, evaluate hierarchical constraints, forward events, update the interface and the particle system. Entities extend a common superclass and can have zero or more meshes and collision boxes attached to them.
- Complex objects: Meshes were created and textured in Maya and exported with the COLLADA format. The xml files are read by the AssImp library and converted into our own data structures. Tangents and bitangents are computed by our own code.
- Animated objects: We use bone animation to animate our objects. The joint transforms are read from the COLLADA files. The joint matrices are updated every frame on the CPU. One vertex can have up to two joints attached to it and is transformed into bone space on the GPU.
- View Frustum Culling: The culling is done by checking every frame for every entity if its bounding box is inside of or intersects with the camera's frustum.
- Transparency: We use transparency for particle effects (as can be seen when hitting an enemy or casting a spell) and for the mark on the ground when casting a spell. The particles are depth-sorted every frame before drawing them.

- Experimenting with OpenGL: We use VBOs and a VAO for every mesh, as well as an FBO to achieve the silhouette effect around items when the mouse is hovering above them. The framerate display in the console will be displayed upon pressing F2. Wire frame mode can be switched on and off with F3 (some of the models are so detailed that the wireframe mode doesn't make a difference though...). Texture quality and mipmapping quality can be changed with F4 and F5. F8 switches frustum culling, F9 transparency.
- Lights: the game uses omni lights to light the static geometry. Each part of the geometry knows the lights which are close and pushes those into the shader. Dynamic meshes (like the player) are lit by the light sources which are close to the camera.
- External resources: flextGL, glfw, glm, AssImp, a few textures from the TotalTextures repository
- Tools: Maya, Visual Studio 2010

Effects

- Bone animations with vertex skinning: The weights for each vertex are exported with COLLADA and read in by AssImp. The transforms for each vertex are hierarchically multiplied. They are then loaded into the shader as uniforms. Each vertex has attributes for weights and for the index of the according joint. The position of the vertex is interpolated between the transformations with the two joints.
- Normal mapping: The tangents and bitangents are calculated by our own code when the mesh is loaded from the COLLADA file. Then they can be serialized so that they don't have to be calculated every time the game is started. We used http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/ as a reference for the calculations and for transformation into tangent space. (Until the presentation we will add complex objects to the scene in order to show off the effect.)
- Dilation shader: The item under the mouse cursor is painted uniformly white into a framebuffer. Then we use dilation with a circular bitmask the extend the area and subtract the original white pixels to get the silhouette.