Runewarrior Arena – Documentation

by
Luca Maestri
Zeno Casellato

Gameplay:

Game:
The objective of the game is to **defeat all monsters** attacking the player-character. There is a limited number of monsters spawning over time, so once they are all defeated the game is won. Contrarily if the the enemies get close to the player too often, he loses all his life and has lost (If the player is hit the screen will flash red). To beat the game the player has **4 abilities** at his disposal. At the first activation a rune is placed at the players position and upon the second activation the ability is triggered. The first 3 abilities, when triggered will **kill monsters of the same color** standing on the rune. The 4th ability is a **teleportation** skill: when triggered the player will be teleported to the symbol. Lastly if the player gets surrounded by foes he can **jump** over them.

Camera controls:
The camera follows the player-character as he moves. This feature can be switched off and back on by pressing '**C**'. With the **arrow keys** the camera can be moved and rotated with the numpad keys '**8**', '**6**', '**4**' and '**2**'. If the player decides to reset the view to its original position on the character then he just needs to press '**X**'.

Player controls:
The Character is moved using '**W**' and '**S**' to move forward and backwards and '**A**' and '**D**' to rotate. He has certain abilities at his disposal to beat the game. He may place certain runes on the ground at his position by pressing '**1**', '**2**', '**3**' or '**E**' as suggested by the interface. By pressing these keys again the runes are activated. If the Player needs to see a brief tutorial on how the game works he needs only press '**P**', which also pauses the game. The player may jump over his foes, thus avoiding unit collision, by pressing the **SPACEBAR**.

Interface:
The very simple interface shows an icon for each ability. When these abilities are used for the first time (placing the rune on the ground), the icon will change to show that the rune is ready to be triggered. Once triggered no icon will be displayed until the rune can be cast again. Also if the player is hit the entire screen will flash red to alert the player.

Extra:
The player may choose to turn the cell shading effect on and off by pressing '**F1**'.

Effects:

Shadowmaps:

(Shadowshader)

To cast the shadows on the ground we used shadowmaps. For this first we had to render the scene from light perspective into a framebufferobject with a texture binded to his depthbuffer. The texture is then used to control wich vertex is in a shadow an wich is not. To eliminate selfshadowing we deactivated shadows for the objects but added code to the shader to darken the faces looking away from the camera getting to the final result.

Cell shading:

(Postprocessshader)

For cell shading we had to render the scene with shadowmaps to another framebufferobject. This time the texture was bound to his colorbuffer to get the scene as it is on a 2D texture. Now all we had to do was to write a postprocess shader that applies the texture on a plane covering all of the screen in the vertexshader and applying a sobelfilter on the texture in the fragmentshader. Mixing the result of the sobelfilter with the original texture results in the scene as the player sees it.

Particle System:
The particles are used for the runes. When a rune is placed on the ground, particles will start coming out of it until the player activates the rune. When the rune is activated again after being placed on the ground it will explode and the particles will shoot in all directions. The particles just a mere plane with an transparent texture on it, always turned to the camera giving the impression of being a 3D object.

Illumination:

(Modelshader)

For the illumination we created a light class containing all the lights components (ambient, diffuse, specular, position). When drawing an object the MVP Matrix, MV Matrix and NormalMatrix are calculated and send as uniform to the shaders. The vertices, normals and uv-mappings are stored in vertexbufferobjects and also send to the shader. In the vertexshader we calculate the vertex position, normal vector and light position in view space and the vertex positionin MVP-space (pixel position on screen), The normal vector, the uv-mappings and the lightposition are normalized, interpolated and sent to the fragmentshader. Finally in the fragment shader the ambient, diffuse and specular light are calculated from the N, L and R vector. This components are added to the Texture-color read from the loaded texture with the texture 2D method and the sampler2D. So a simple Phong-model is created.

Animated Objects:

(Modelshader)

For the animations we mainly used Blender to create an armature for the model and finally animate the model. The created animation is then exported into an directX file, so that the assimp library can import the model and animation into our project. In our project we used the animator class from the assimp development team. The animator class interpolates the keyframes saved into the directX files and calculates the bone matrices for the animation. Also for the animations to work we had to render the models using the assimp nodes and data structures. In the vertexshader the position of each vertex is transformed with the given bone matrices to simulate the movement of the model. Each model has one animation created with blender (walking animations) and one simpler animation created manually with transformation in our project (monster dying animation and player teleport animation).

Experimenting with OpenGL:

- The cell shading effect can be turned on and off by pressing the F1 key. It switches between methods used by the post-process fragmentshader.
- The framerate is displayed in the window title.

Features:

- Skeleton animations
- Nice graphics
- Different particle systems
- Smooth movement
- Free movable camera
- Camera can be centered or locked on player
- Diffrent monster types
- Interface
- Tutorial screen / pause
- 4 skills
- 4 different spawn points for monsters
- Random calculations for monster spawn. It will never be the same!

Tools & libraries:

Tools:

- Blender, for 3D models and animations
- Visual Studio C++ 2010 Express
- gDEBugger
- AssimpViewer

additional libraries:

- glfw http://www.glfw.org/
- sdl http://www.libsdl.org/
- glew http://glew.sourceforge.net/
- glm http://glm.g-truc.net/
- assimp http://assimp.sourceforge.net/

Sources:

- Postprocessing: www.geeks3d.com (18.06.2012)
- Skybox: http://ogldev.atspace.co.uk/index.html (18.06.2012)
- Assimp: http://ogldev.atspace.co.uk/www/tutorial22/tutorial22.html (18.06.2012)
- Shadowmaps: http://ogldev.atspace.co.uk/www/tutorial23/tutorial23.html (18.06.2012)
- Lighting: http://www.lighthouse3d.com/ (18.06.2012)