SteelWorms

Adam Papp

16.06.2014

1 Starting the game

The game can be started in full screen mode with 60Hz refresh rate by double clicking on the exe. If it is started from command line, it must be started from the bin directory. In this case a dummy argument like *windowed* can be used to start in windowed mode. The current version is only a two-player version, since there is no menu to set the number of players, but the algorithm can handle any number of player. Please note, that the shader and texture folder are placed next to the bin folder in the zip file. This is done, because the development environment is also set up this way and the exe finds files using the following structure:

- {any directory}/SteelWorms.exe
- shader/shadow.vert
- texture/heightmap.bmp

2 Implementation

The implementation consist of the following classes:

- Texture: Encapsulates a texture in the GPU. It holds the handle for the texture in the GPU memory. The BMP loading algorithm is a static function of the texture class.

- Model: Encapsulates a graphical object model in the GPU. Holds the GPU vertex buffer objects for indices, vertices, normals, texture coordinates. This way the every model is only once in the GPU memory.
- Shader: Encapsulates a shader program. It holds the handle for the GPU Shader program.
- Scene: A helper class to be able to access such objects like the terrain and camera from other SceneObjects. It holds all the SceneObjects which are rendered.
- SceneObject: The base class of the objects which are rendered. It has a pointer to a shader, a model, a scene, holds the model matrix and a handle to a vertex array object. It has a parent child hierarchy. The drawing function for all the SceneObjects are written here. Uses the *generic* vertex and fragment shaders.
- Sun: Implements the shadow map creation. Returns constants for lighting. Uses the *shadow* vertex and fragment shaders.
- Cube: Obsolete.
- Camera: Implements the camera movement. It is inherited from the SceneObject.
- Terrain: Computes the height of the terrain at any position. To create a water surface also this class is used. It is inherited from the SceneObject.
- Tank: Implements the tank movement. The parent of the Turret class which is the parent of the Gun class. They are inherited from the SceneObject.
- WheelTrack: Implements a CPU particle system. It holds an own Model of one quad. Uses the *particle* vertex and fragment shaders.
- Bullet: Implements the bullet movement. It is inherited from the SceneObject.

- GameLogic: The complete game logic is written in this class. It has pointers to the player tanks. It is responsible for handling turns, creating and destroying the bullet, to detect bullet impact, to detect water damage and for the win/loose condition.
- Debug: The switches to experiment OpenGL performance and quality are implemented here.

2.1 Camera

The camera is a SceneObject, but it has also a view matrix. The computation of the view matrix is done from the horizontal and vertical angles of the camera. From these Spherical coordinates the Cartesian coordinates are computed, these are the *direction* and the *right* vector. The cross product of this two vector gives the *up* vector. Then using the glm::lookAt function, the view matrix is computed. The camera has a projection matrix. On zooming this projection matrix is recomputed using the glm::perspective function. When an object is being rendered, the view matrix of the camera is returned. The following website was used to understand and implement the camera concept. *http://www.opengl-tutorial.org/beginners-tutorials/tutorial-6-keyboard-and-mouse/.*

2.2 Animated objects

The tanks are moved by user interaction. They can be moved forward or backward and can be rotated. A tank moves on the terrain. Every tank has a turret, which can me rotated right and left. The turret has gun, which can be tilted up and down. They are connected with a parent child relation. The tank model was found online(see License in texture folder). To import it, the Assimp library was used.

Important: The latest Assimp library must be compiled from the repository. The current(3.0) version fails to load the model. The texture coordinates are still loaded incorrectly. Probably this happens because the model was generated using a very old program. Also there is something wrong with the normal vectors. One track of the model is upside down. The reason for the flickering on the side is unknown. The new version(3.1.1) crashes. Maybe the problem comes from the MSVC10 compiler. In the SteelWorms repository the uploaded Assimp library was compiled using an MSVC10 compiler.

2.3 Controls

The user inputs are handled by the GLFW library. The camera can be moved either with the keyboard or with the mouse. The keyboard controls are the arrows for moving and 1 to zoom out and 2 to zoom in. The right mouse button can be used to move the camera, the left mouse button can be used to rotate the camera and the middle mouse button is to zoom. A tank can be moved with the WASD buttons. The turret and the gun can be moved by the IJKL buttons. A bullet can be shot with the SPACE button, the power of the shoot depends on the amount of SPACE. A turn can be ended with the ENTER key. The ESC key can be used to exit the game.

2.4 Gameplay

This version is a two player version. In one turn only one tank can be moved. A turn ends in one minute or when the bullet reaches the terrain. After the bullet is shot, the tank can not move. A bullet can be shot by holding the SPACE key. The longer it is hold the bigger power it will be used to shoot. There is water under and above the track. If a tank reaches water it's health will decrease. If a tank goes off track it dies. When only one tank lives, the game ends and the result is written out in the console.

2.5 Transparency

To achieve transparency, a water surface is constructed. The Terrain class is used to display the water. It is textured with 0.5 transparency. Before the water, the tank trail is rendered. This is done to see the track under the water. The track's transparency is decreased with time until it reaches zero.

3 Effects

3.1 Shadow map

The only light source is the Sun. It is located at the middle of the terrain. It does not produce too much light to be able to test the lighting. The following website was used to understand and implement the lighting. *http://www.opengl-tutorial.org/beginners-tutorials/tutorial-8-basic-shading/*

The light for the shadow is defined as a directional light source. (Note: This makes it strange, that the light is decreasing to the borders, but the shadow is not getting longer. A perspective matrix should be used instead of orthogonal.) The scene is being rendered from the light source into a Frame-Buffer, which has a Texture attached to it. The texture will contain at each position the distance from the light source. When rendering the scene using the camera, the same distance is computed and compared with the one in the texture. If it is less, then the fragment is in shadow. To improve the quality of the shadows, PCF is being used. This is achived by using sampler2Dshadow in the fragment shader. The comparison must be enabled in texture parameters. To requesting a value from the sampler2Dshadow, the x and y texture coordinates and the light distance in the z are given. The sampler will compare also neighboring texels and gives back a value between 0 and 1, the ratio of the texels, which were less then the requested z. The following website was used to understand and implement the shadow maps.*http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/*

3.2 Particle system - CPU

When the tanks are moving, they are leaving trails after themselves. This is done by a CPU sided particle system. Each trail is a simple quad. On initialization this quad is uploaded to the GPU memory only once. When a new trail is added, it's position is written into a buffer and it's transparency is 1. During each update, the transparency is decreased with a factor until it reaches 0. On zero the particle is dead and not updated. On drawing the position and the transparency of each living particle is uploaded and drawn with one call. The following website was used to understand and implement the particle system. *http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/*

4 Features

4.1 Terrain

The heightmap is loaded from a grayscale image, where each pixel represents the height. The same BMP loading algorithm is used, which loads textures. The image must not have padding bytes at the end of the rows. A triangle is

built up from 3 pixels, so 4 neighboring pixel represents two triangles. The pixel values(scaled to 0..1) are stored in a lookup table. With the function getHeight(x, y) it is possible to get the height value for any x and y coordinate. The idea of the algorithm which computes the heights is from the "Introduction to 3D game programming with DirectX 9.0" book page 228-231 written by "Rod Lopez, Frank D. Luna". The algorithm computes the position of the 4 pixel values surrounded by the given x and y coordinate and then interpolates the height value from the corresponding coordinates.

4.2 Tank

Each player controls one tank. The color of the tank shows the health of a tank. By controlling the turret and the gun the shooting direction and angle can be modified. Holding the space longer will make the bullet to fly further. The closer the bullet reaches the center of an other tank the more damage it does. The health of the tanks will decrease if they are in standing in water in they current turn. The deeper the water the more they loose health.

4.3 Bullet

The trajectory of a bullet is computed by the following equation, which was found on Wikipedia(Ballistic_trajectory).

$z = z_0 + d\tan\theta - \frac{gd^2}{2(v\cos\theta)^2}$

This equation computes the height of the bullet. The value v in the equation is the velocity. The longer the SPACE key is held to shoot, the higher this value will be. The value d is the distance from the shoot point. Theta is the angle, which is currently 45 degrees. Value g is the gravity, set to 9.81.

The collision is detected by comparing the height of the bullet with the height of the terrain at the x and y coordinate of the bullet. If the difference is negative, then the bullet has reached the ground. Then the distance of the impact point and all the tanks are compared. If the distance is under a threshold, then the health of the actual tank is decreased according to the distance.

4.4 Illumination and textures

All the rendered objects, except the tank trails are being illuminated by the Sun. The bullet is modeled with a sphere and textured. The water is modeled with a heightmap and textured. Only BMP textures can be loaded, because no external library is used for images. The specification can be found on Wikipedia(BMP file format). This algorithm loads also the padding bytes at the end of each row.

5 Experiment

The frame time can be turned on with F2, wireframe can be toggled with F3 and F9 turns off transparency. The sampling quality of the textures can be switched using F4 and the mipmapping can be switched using F5. Note: The mipmaps are computed even if they are switched off, but they are not used. The shadows with the linear filtering have a better and smoother result. The PCF only works when using linear filtering. The water surface is rendered faster using mipmapping. When using mipmapping the render time is around 2.7 ms, without mipmap using nearest 3.2 ms, using linear 3.5 ms. Shooting a bullet can decrease to 4.4 ms, because of the high polygon count of a sphere.

6 Additional libraries

- GLFW - An OpenGL library http://www.glfw.org/
- GLEW - The OpenGL Extension Wrangler Library http://glew.sourceforge.net/
- GLM - OpenGL Mathematics http://glm.g-truc.net/
- Assimp - Model loader http://assimp.sourceforge.net/ Note: See Animated Objects