Developing a Snow Board Game

Results of a practical course at the Chair for Computer Graphics and Multimedia
(RWTH Aachen University, Germany)

Melvin Bender* Julian Steinsberger-Duehrssen† Dominick Holman‡ Deniz Kesmez§ Lavinia Goldermann¶

Figure 1: *game teaser*

Abstract

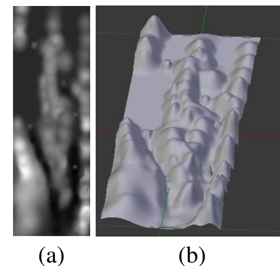
The task of this practical course was the development of a game for the IPhone 6 and the Dive Glasses. We decided to develop a snowbording game, controled only by head movement. The goal is to get as much points as possible in the given time by collecting coins and power ups and avoiding obstacles like trees.

Keywords: game programming, Snow Board Game

1 Content Creation

For a snow board simulator a lot of different 3D models are needed. The most important object that needs to be created for the game is the snow board course itself. To precisely place textures and objects on to the map, height maps were used to generate the maps inside the OpenGL environment. Height maps are image files that represent an 3D object by assigning different shades of grey to heights. The course itself was created by using the Blender software and adjusting the properties of the course in a way, that the colour of the resulting height map represents the heights of the map. Then the height map was produced by placing the camera above the map The other objects like trees or snow men were also created with the tools of the blender software. To guarantee a good performance of the final game, only simple geometry was used for the different object models. The geometry of the objects was put into the game by exporting the blender files into Wavefront obj. files. Texturing was done by hand with the brush tool of the blender software. Since the textures are not handled by the obj. files, exporting the texture was done by saving an image provided by the Blender software.

*melvin.bender@rwth-aachen.de
†johannes.steinsberger-duehrssen@rwth-aachen.de
‡dominick.holman@rwth-aachen.de
§deniz.kesmez@rwth-aachen.de
¶lavinia.goldermann@rwth-aachen.de



(a) (b)

Figure 2: *map as height map and in Blender*

2 Map Loading

For loading the maps the data of the provided height maps is processed and heights are calculated. In order to do that, the RGB values of the pixels of the image are mapped linearly onto height values between zero and a defined maximum height. These heights are then used as the z value of the vertices in a vertex grid with the same relation of the length and wide as the height maps. Further more the vertex normals and texture coordinates are calculated using the x y and z values of the vertices.
For rendering the map an indexed vertex buffer is used, so every vertex has to be put into the buffer only once. For this an index buffer is generated in which three consecutive indices, that reference three vertices in the vertex buffer, represent one triangle, see [Ind May, 2012].
Additionally to the height map two more images are used for map generation, a blend map and a entity map.
The blend map is used on the one side as an input to a vertex shader for drawing the textures and on the other side for storing the information wich texture is present at a certain vertex. This makes area detection possible.
The information of the entity map will be used to put the different entities on the map and for storing the positions at wich an object is present. This information will be used for hit detection.

There are two vertex buffer used so that two maps can be loaded at

the same time. When passing half the length of a map the next map will be loaded into the buffer. There are two modes which can be used for selecting the next map. Cyclic map switching always picks the next map form a list of maps provided in a txt file and starts again at the beginning if the end is reached. Random map switching selects the next map randomly from this list. At the border of two maps a Gaussian filter is used to smoothen the connection. Additionally the vertices at the connection of the two maps are put at the same height. This results in the necessity of recalculating the normals in this area.
This approach is used, rather than loading all required maps at the initialisation of the game, to enable an endless track and reduce the memory usage. For this switching process a second thread is used as this can take up to half a second depending on the device.

3 Movement

The movement of the game consists of two parts. The basic movement dependent on the underground and the hit detection.

3.1 Movement / Area detection

For moving the player a movement vector is used which points in the derection in which the player is moving. This vector is rotated around the y-axis by the pitch value of the device attitude.
The camera representing the player will then be moved by a certain speed every frame into the direction this vector is pointing. With the information retrieved from the blend map on for the underground the speed will increase or decrease. On snow for example the player first will increase its speed fast and later the acceleration will decrease an the speed will approach a maximum value.

3.2 Hit Detection

For hit detection every type of entity used gets a bounding box. Then every frame an intersection of the line of movement with a bounding box of an entity in a certain area will be searched. For testing if the lines intersects with one of the sides of a bounding box or if it lies inside it an algorithm based on the method described in [Lin August, 2015] is used.
When a hit is detected an action depending on the object hit will be performed.

4 Graphics

4.1 Management

The first important part of painting the world on to the screen, is to separate the scene by elements which are in front of the camera and which aren't. At first we are calculating the view frustum from the position of the camera. With that information we are painting a box around the frustum, so we only need to look in the area from the box in our table if there is an object and if it is in the frustum.
We are managing a list of all the objects with these characteristics. This list will be edited in the rendering process, so that each object gets loaded once with all informations painted on every position, before the next object gets loaded.

4.2 Graphical Effects

4.2.1 Multiple-Light-Sources

Our main light source is the sun/moon, which is moving the whole time and effects the lighting of the models and world. For a day/night cycle this light source is constantly changing colours. We use Phong Lighting to implement the lighing with separate settings for each object for ambient, diffuse and specular color and a shininess value for the specular Light. For faster calculations we worked with the *Phong-Blinn-Approximation*.
Specular light and point lights are only calculated near the camera for speeding up the render process. The the torches are point lights and the effect is calculated by the distance to the verteces of objects and the world.

4.2.2 Fog

Objects and the world will disappear in fog as they get harther away form the camera .

4.2.3 Skymap

With cube mapping and 6 textures we are calculating a sky. Two sets which are blending in each other we are getting the day night cycle. To simulate sky movement the textures are rotating.

4.2.4 Reflection

On some game-models like coins and power ups and on the ice texture of the mapn the reflection of the sky is calculated by calculating the corresponding area in the sky map. Like with lights reflections are only calculated near the camera.

4.2.5 Shadow

For casting shadows we use shadow maping. For improving the quality of the shadows the shadow map is multisampled and additionally Poisson sampling is used for smothing the edges of the shadows. Again this calculations are only done near to the camera.

4.2.6 Blur Effect

A blur effect is simulated depending on the speed the camera is moving with. This effect is only calculated for the right and left side of the screen and gives the player a better feeling for the speed. At last this effect has to seperated for the split view.

4.2.7 Particular Effect

We are using a particular effect on two occasions. For a snow effect a texture gets rendered at positions definde by a sinus funktion. For the fire effect of the torches the fire particles have three diffrent stages with different colours: smoke in grey, fire in orange and when igniting in yellow. When geting higher these particles get smaler.

5 Sound

We use the *AV Audio Player* to play multiple sounds at a time. Every sound gets its own sound buffer.

References

May, 2012. Android lesson eight: An introduction to index buffer objects. http://www.learnopengles.com/android-lesson-eight-an-introduction-to-index-buffer-objects-ibos/.

August, 2015. Elementare geometrische methoden: Schnitt von strecken. http://www.imn.htwk-leipzig.de/ medocpro/buecher/sedge1/k24t3.html.