Virtual Slingshot Practice Ground

Moritz Laass[*]

Abstract

Coming up with compelling Virtual Reality Experience is a difficult task and even though the medium has been around for more than two decades it is still in its infancy. This Document will try to give insight into the development of a Virtual Reality Practice Ground application, that is designed to run in a CAVE environment. We will look at what was done, the design considerations, the techniques that were employed. Some implementation details and also what could be done to further improve the experience.

1 Introduction

Slingshots have been a popular toy and weapon for a long time. In this project a virtual slingshot training ground was implemented for a CAVE environment. The user has the experience of holding a slingshot and shooting wirtual projectiles at a target. Hitting the target is detected and a message appears on the counting station indicating the color that has been hit on the target and the number of points this generates. Through subsequent hitting of the target the accumulated points are visualized in the risingg of a marker on the point counter. This documentation will explain the user interface, give an overview of the different components of the programm, how they work togther and also look at implementation details like physics and trigonometry.

2 User Interface

The programm is controlled using two tracking devices, that are held in both hands. The left hand holds the passive device which represents the slingshot itself. The Right hand uses the wand device, which is used to pull the slingshot sling and aim. Also the Analog stick on top of the wand is used to adjust the target distance.

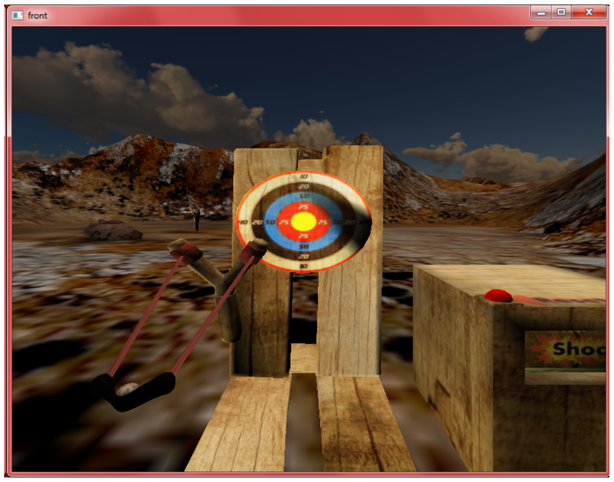

Figure 1: Shooting in Action.

[*]e-mail: moritz.laass@gmail.com

2.1 Slingshot Action

To shoot a projectile the user has to bring the right hand holding the wand close enough to the left hand holding the slingshot and press the trigger button. If this is done correctly, the projectile is put into the pocket of the slingshot and the pocket ”sticks” to the right hand and the rubber strings of the slingshot stretch accordingly. The user can then adjust his position and aim for the target. When the trigger button is released the projectile is proppelled in the correct direction and the rubber bands try to move back into their original position using a elastic relaxation technique.

2.2 The Score Counter

The score counter was added to give additional incentive to play and to give feedback of the user’s performance. It consist of a box with a wooden texture that has a front window to the ”display” a red reset button and a pole that is used to show accumulated points. The display is realized using an eigth sided cylinder, each side contains one of the possible display messages. To change the display the cylinder is rotated by $45° \times index$, where $index$ is a value between $0-7$ and indicates the side that should be shown.

When the user starts aiming the display changes to the ”shoot” surface $index = 0$. When the target is hit the dsiplay changes to the the side that corresponds to the color of the ring that was hit on the target, also the socr egets updated and the marker on the score pole gets translated accordingly. When the target is missed, the display shows ”Missed”. Once the Score reaches its upper limit the display chanegs to ”Full Score” $index = 7$.

At any point the user can take the wand close to the big red ”Reset” button on the corner of the counter and pull the trigger which will reset the score to zero and rotate the display back to the ”shoot” position $index = 0$.

3 Building The Components

All of the 3d components where build in blender and the textures are taken from public sources. All models where exported as .obj and accompanying .mtl files with a scaling factor of one hundred, which means one unit length in blender becomes hundred unit lengths in the CAVE. This was done because one unit in the CAVE is equal to one centimeter whereas for modeling it seemed more convenient to think of one unit as one meter.

3.1 The Environment

The Landscape was generated using a blender plugin. in the next step rocks where randomly distributed across the surface using a ”hair” particle system. The landscape and the rocks were textured using the same rocky texture and using blenders cycles render engine light and shadows were ”baked” into the texture for the rocks and the landscape to add to the realism of the scene. A t dead tree was modeled in blender using subdivision surfaces and placed in sight of the player to add a unique conterpoint to the environment.

3.2 The Skybox

The Was also modeled in blender, whats special about it is the fact that the normals point to the inside of the box. Also the Bottom of the skybox was left out as the landscape covers it. The Textures where taken from an excellent free skybox set.

3.3 The Target

The target is comprised of two components, the actual target and the target stand that looks like it was build using rough woodplanks, this was done to add to the realism, by not having the target float midair, but give the impression that the person is actually in a real environment. by splitting up the target and the stand the target height above ground can be adjusted, which was useful to figure out the ideal position of the target.

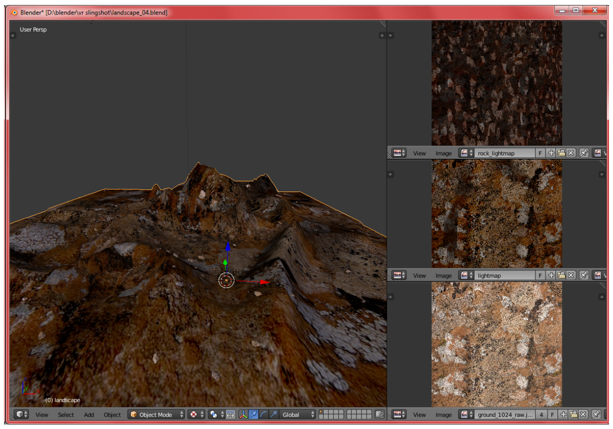

Figure 2: Building the landscape on the left, different textures and lightmaps on the right

4 Math & Physics

No Simulation is complete without some interesting math or physics problem. In the case of this project the interesting parts are the simulation of the slingshot strings, the

4.1 Wobbly Slingshot Strings

The strings of the slingshot are modelled using two common cylinders with a simple texture. The cylinders are the resized and rotated to simpulate the behaviour of real slingshot strings. The following listing shows the Calculation of rotation and the scaling of the left string.

```
//distance vector between slingshot pivot
//and pocket anchor point
Vec3f d0 = pocketPoints[0] - slingPoints[0];

float l0 = d0.length(); //length

//scale along the y axis
stringScale[0][1] = l0/20.0f;

//thickness of the string: longer = thinner
float thk0 = 0.5 + 5* 1/l0;
stringScale[0] = Vec3f(thk0, l0/20.f, thk0);

//calculate angle (dot) and
//axis (cross) for the rotation
d0.normalize();
float ang0 = acosf(Vec3f(0,1,0).dot(d0));
Vec3f ax0 = Vec3f(0,1,0).cross(d0);

//generate rotation quaternion
stringRot[1] = Quaternion(ax1, ang1);
```

The bounciness is created by incrementally trying to reach a desired relaxation point (slingPoints[SL_LOOSE]) by adjusting a change speed and adding that speed to the actual position.

```
void calcStringFollow(float dt) {
  Vec3f d = slingPoints[SL_LOOSE] - stringFollow;
  stringFollow += followSpeed * dt * 240.0f;
  d = d * 0.8;
  followSpeed = (followSpeed *43.f *dt) + d*dt;
}
```

The numeric values in the above listing have been experimentally adjusted to give a nice impression of semi-realism and still maintaining a stable simulation.

4.2 Flight of the Projectile

The projectile gets a speed vector that depends linearly on the distance between the the right hand and the left hand upon trigger release. Over time gravity is added to simulate a realitic trajectory

```
Vec3f gravity = Vec3f(0.0f, -981.f, 0.0f);
projSpeed += gravity * dt * fact;
//new position
Vec3f nPos = projPos + (projSpeed * dt * fact) ;
```

Calculating if the target is hit happens when the projectile crosses the ”horizon” of the target on the z-axis. The intersection point is then calculated and if it is close enough to the target center, the score and display get updated accordingly.

```
if (targetPos[2] < projPos[2] && targetPos[2] > nPos[2]
  Vec3f d = nPos - projPos;
  float t = (targetPos[2] - projPos[2]) / d[2];
  //calc the intersection point:
  Vec3f inter = projPos + (d*t);

  Vec3f d2 = targetPos - inter;
  float dist = d2.length();
  if(dist <= targetSize){
    hitTarget(inter)
  }
```

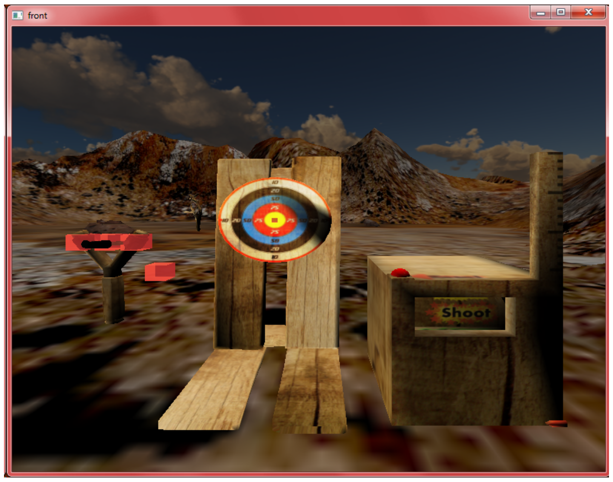

Figure 3: The scene with transparent ”helper” boxes that were used to visualize positions during development.

5 Conclusion

in conclusion the project was very straight forward to implement, the bulk load of the work was done on a windows machine. Compiling and running it in the CAVE environment was flawless, except for one small mistake that did not affect the winows build. The main Challenges were the correct behaviour of the slingshot strings and keeping track of all the different aspects of the scene.