Algorithmic Description of Special Effects

Animated Water Surface

Introduction

The animated water surface is mostly implemented in the vertex shader. This is done via a wave equation which calculates the offset on the y-axis for each vertex and to the new position in respect the normals.

Triangle Mesh

To create a wave, it is needed to create a mesh to manipulate each vertex individually. This mesh is created with an own function which returns all vertices and indices to create it with a TRIANGLE_STRIP call. It also returns normal vectors which are just (0,1,0) to create a default value.

First the vertices are created. This is simply done by iterating over the width and depth and creating a vertex at each point. The indices are created by iterating over each row and add the next index. However, to link the rows together, degenerated triangles are used. This means, that the triangle consists of three vertexes, where two of them are the same. Therefore, the GPU just skips these triangles.

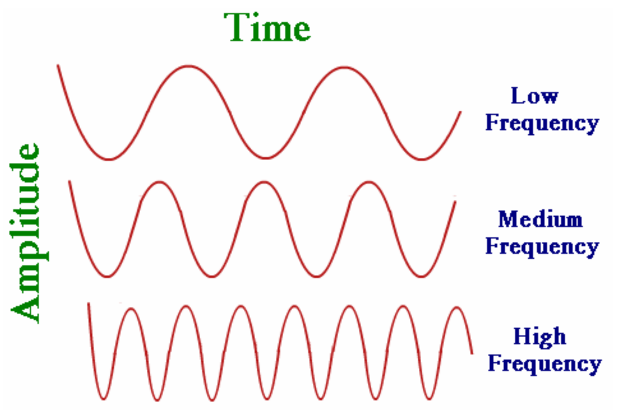
Position/Normal calculation

This mesh is now used in a special vertex shader. In this shader, the y-position of the vertex and the normal vector of each vertex is calculated. For the position, the use of a wave equation is needed. Our wave equation is defined as following:

$$y = \ Amplitude * \sin(Vec_{position} \cdot Vec_{direction} * Frequency + Time * Phase)$$

We can split this equation up to show how it works:

$y = Amplitude * \ldots$
The amplitude defines the maximum and minimum height of the wave. It is used to create different waves with different heights.



$y = \ldots * \sin(\ldots)$
A sine wave is used to create the wave itself.

$y = \ldots * \sin(Vec_{position} \cdot Vec_{direction} * \ldots)$
We use the dot product of the position of the vertex and a fixed direction vector to create the moving part of the wave in respect to a direction. The position is used, because each vertex should have a different offset in respect to its positon.

$y = \ldots * \sin(\ldots * Frequency + \cdots)$
The frequency is used to determine how long each wave (wavelength) should be. This comes in handy, as water has different waves with different frequencies.

$y = \ldots * \sin(\ldots + Time * Phase)$
The addition part takes care of the speed of the animation. The time parameter makes the animation time dependent. This must be used, because the time to render each frame can differ on multiple hardware. In addition, a phase parameter is used to adjust the speed itself.

As the reflection, should also be present, we need to calculate the normal vectors too. To calculate this, we first need the tangent vectors of the plane. To get these, we need to calculate a partial derivate to get the slope of the function in a specific point (vertex position). The partial derivate is nothing more than the derivate which treats every other variable as a constant. We define the partial derivate of x as a and the partial derivate of z as b. Therefore, our tangent vectors are (0, a, 1) and (1, b, 0). To get the normal vector, the cross-product of these is used. As we only need the vector where y=1, the order of this calculation is important. (0, a, 1) x (1, b, 0) = (-b, 1, -a). This normal vector gets now assigned as normal of the vertex.

To get a nice-looking wave, the wave formula is called multiple times with different parameters and added together.

Billboarding

Introduction

The animation of the billboard is done in the AnimationSGNode by setting the billboard matrix with the view matrix.

treeRotate

To get the billboard texture always the face the camera the billboard needs to be rotated the same way as the camera itself.

With the ViewMatrix we can calculate the CameraMatrix as so $CameraMatrix = ViewMatrix^{-1}$

The ViewMatrix contains translation and rotation but we ignore the translation because we only need the rotation.

$$\mathrm{VM} = \begin{array}{c} \begin{array}{cc} \text{rotation} & \text{tranlation} \end{array} \\ \begin{pmatrix} \boxed{\begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{matrix}} & \boxed{\begin{matrix} a_{14} \\ a_{24} \\ a_{34} \end{matrix}} \\ \begin{matrix} a_{41} & a_{42} & a_{43} \end{matrix} & a_{44} \end{pmatrix} \end{array}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}^T = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}$$

The rotation is an orthogonal matrix which means the inverse of the matrix can easily be calculated with the transpose of the matrix (mirror the matrix on its diagonal).

We use the billboarding animation on trees therefore we only want Y Rotation, to avoid that the trees lie flat on the ground if the camera looks down. Therefore, we only need to set the Y rotation of the matrix, which happens in the outer corners of the rotation matrix.

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Now we have the CameraMatrix next step is to set the Y rotation components of the billboard matrix to the camera rotation components (other components are not touched).

To finish off we set the translation components of the billboard to the position of the billboard.

$$\text{ViewMatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \Longrightarrow \text{BillboardMatrix} = \begin{pmatrix} a_{11} & / & a_{31} & \text{b.x} \\ / & / & / & \text{b.y} \\ a_{13} & / & a_{33} & \text{b.z} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$