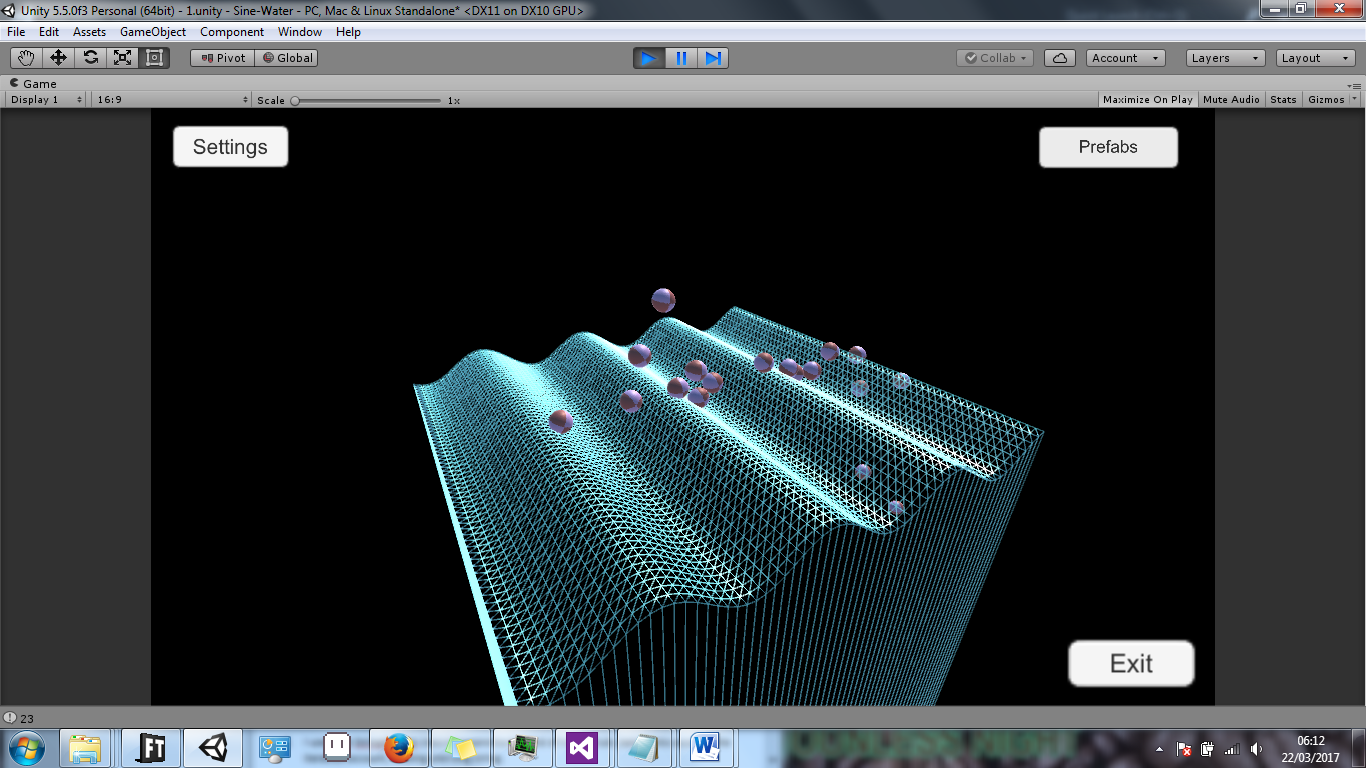
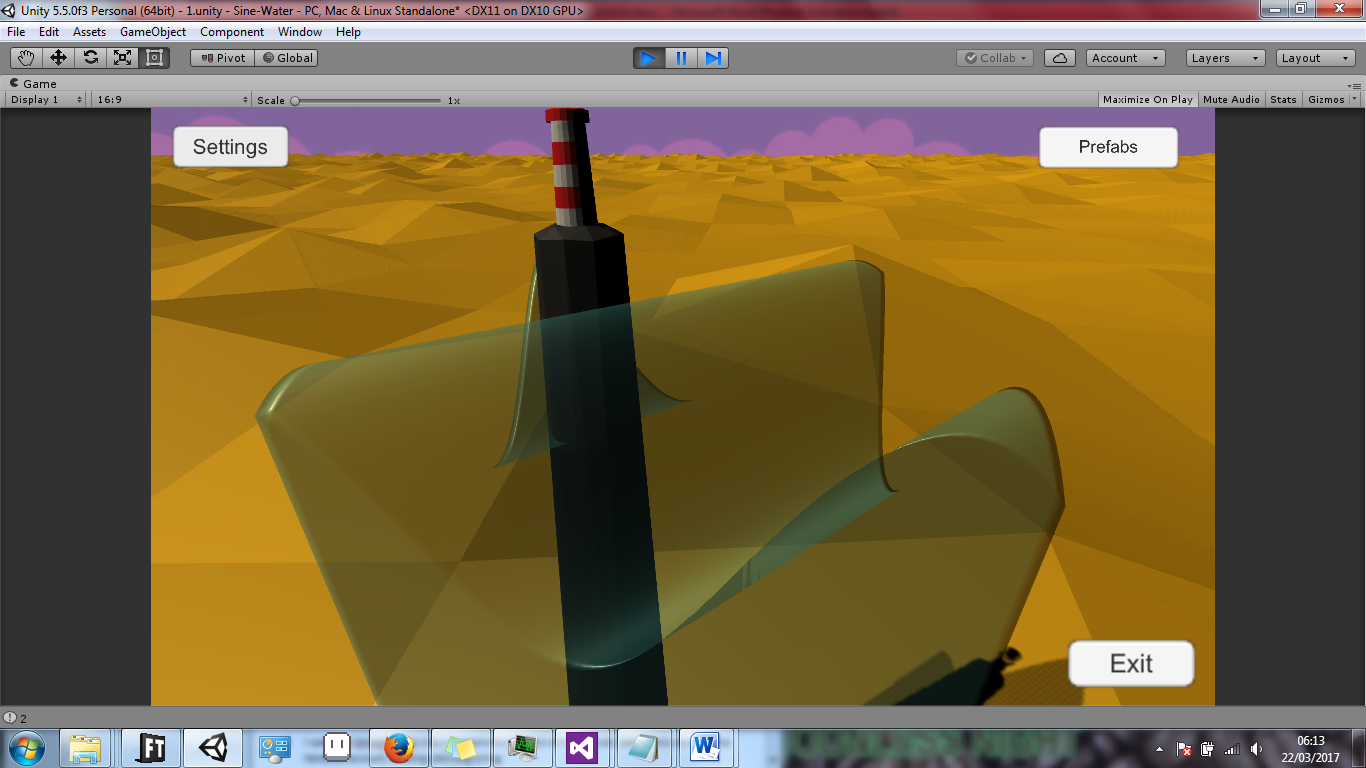
Luke Sanderson

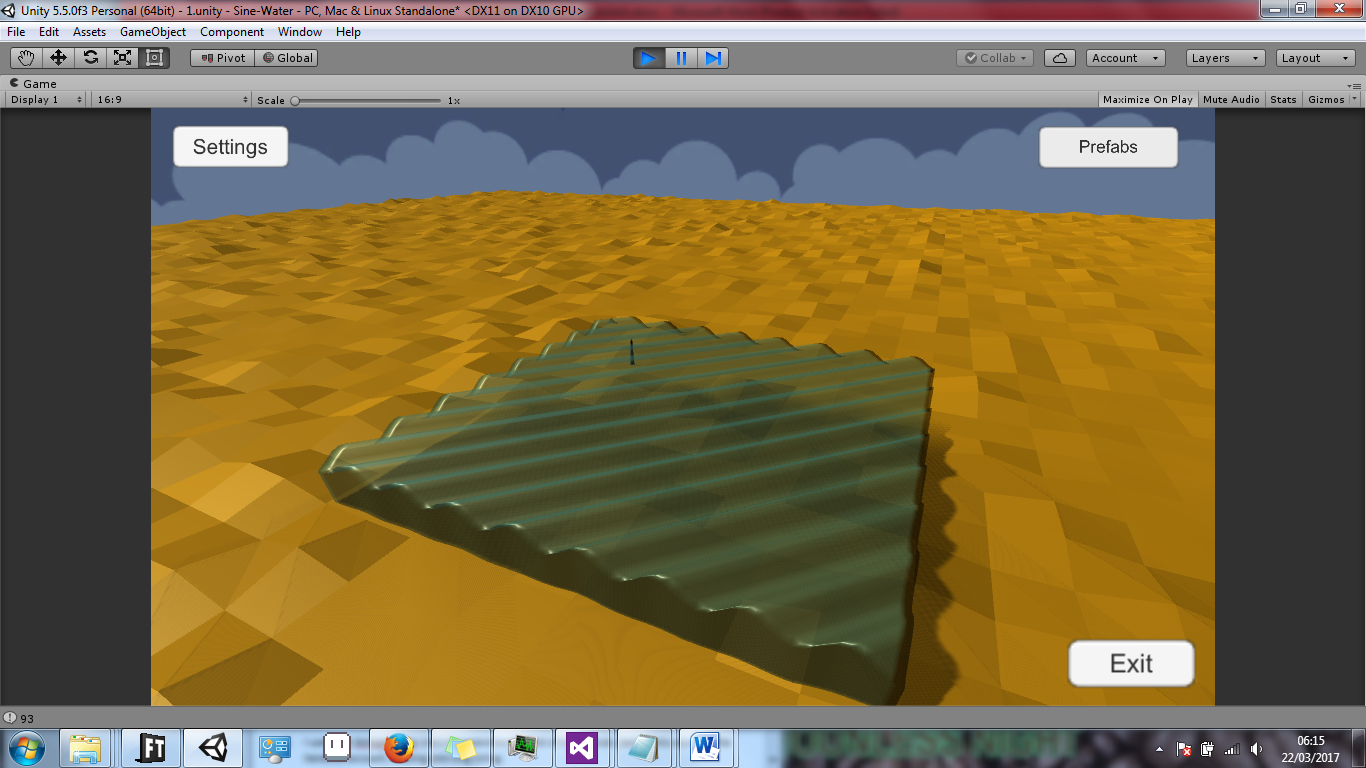
Maths & Graphics 3

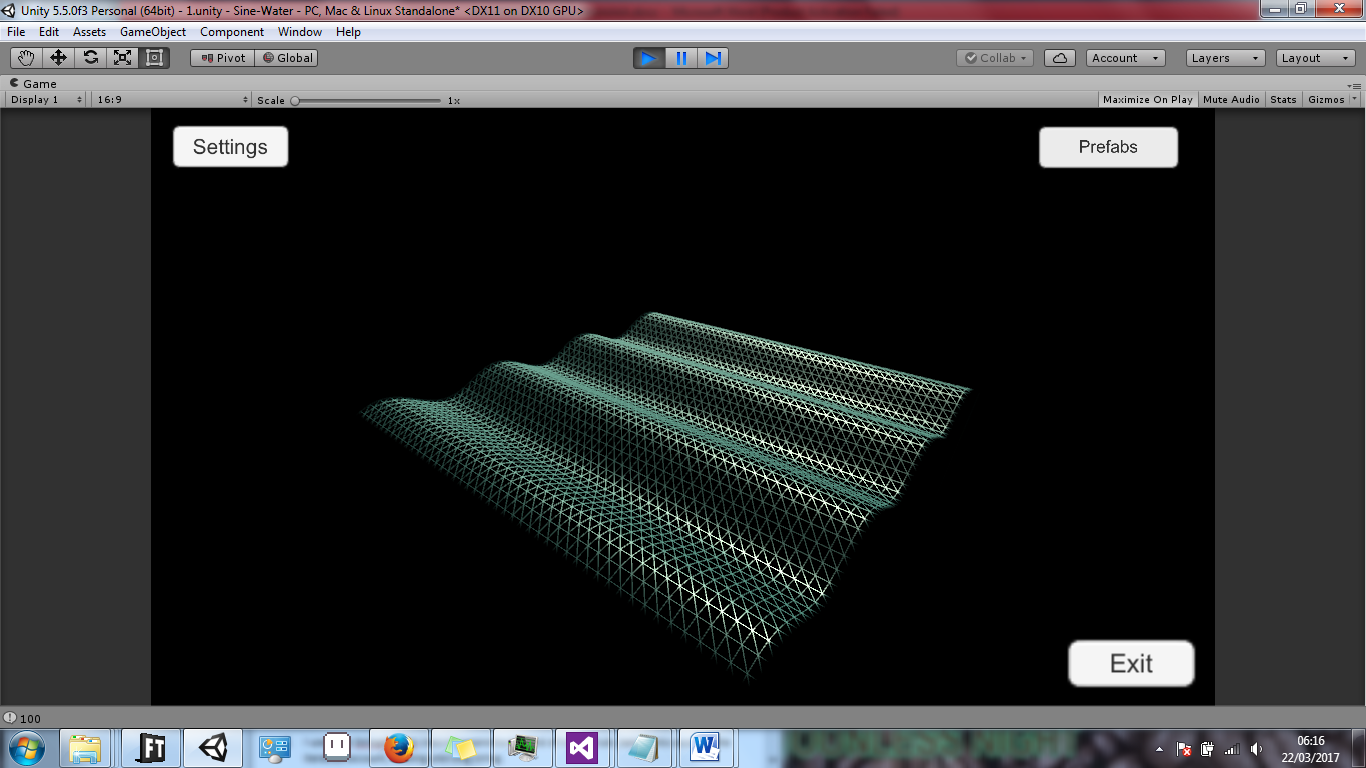
Sine Waves

22/03/2017



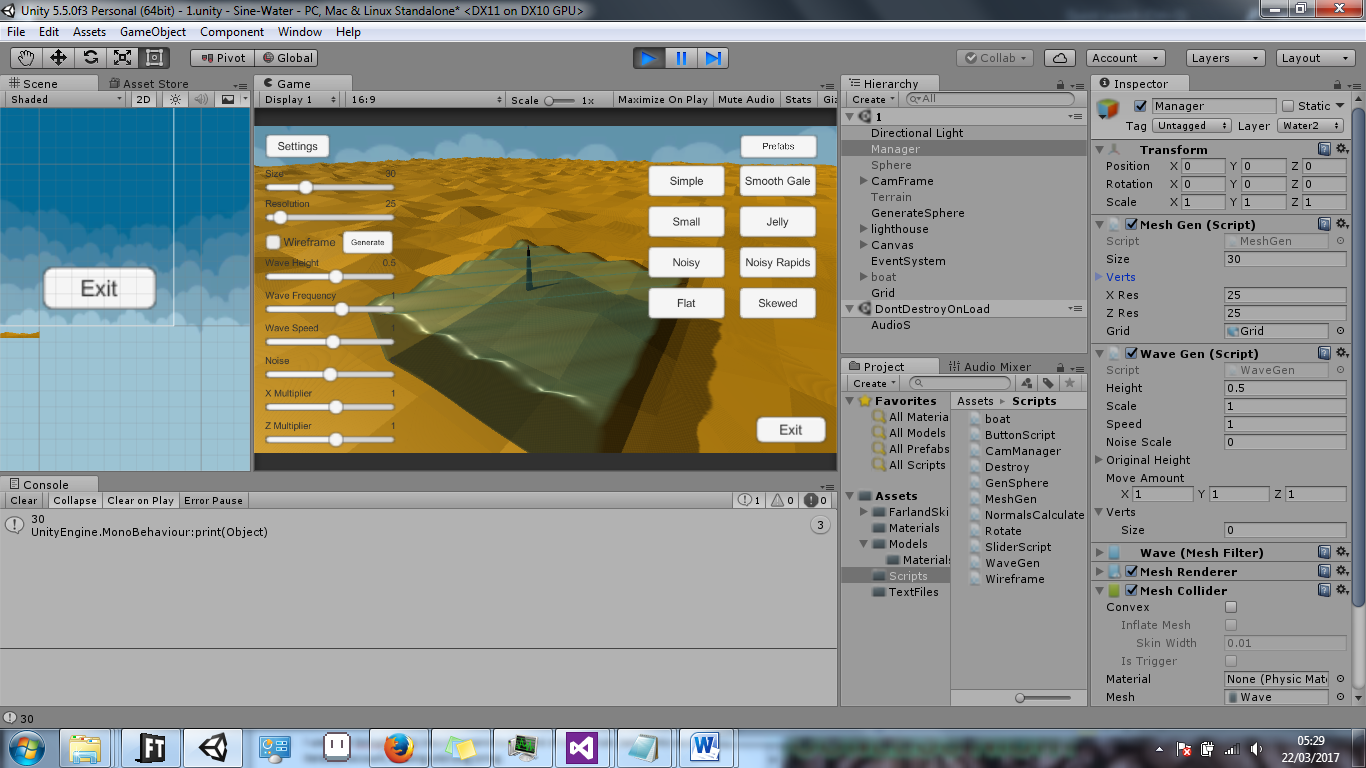






Introduction

*Sine Waves* is a tool for use in Unity3D which allows the user to create a mesh with points that are influenced by a sine wave, thus giving the impression of ‘ocean waves’. The user can load from a number of prefabs which are read from text files, and continue to edit the parameters of the wave further in the in-game editor. I’ve used Unity3D its UI interface easily allows the user to change parameters on the fly, and I have taken extra care in avoiding Unity’s built in primitive plane prefabs.

Code

The majority of the code is run from the *MeshGen.cs* and *WaveGen.cs* scripts, both found in the ‘Manager’ gameObject. The first script creates a mesh of ‘Size’ size (30, in this case), with a resolution of 25 \* 25 vertices. The following is how I created the vertices – I looped through both the x and z resolutions of the wave and created points from these. From these vertices I could calculate where each triangle would be generated, along with their respective normal and UV co-ordinates.

verts = new Vector3[xRes \* zRes]; //Inst vertices

Vector3[] normals = new Vector3[verts.Length]; //Inst normals

Vector2[] uvs = new Vector2[verts.Length]; //Inst uvs

//

// VERTS

//

for (int i = 0; i < xRes; i++) //loop through verts, assign position

{

float xPos = ((float)i / (xRes-1) - 0.5f) \* size;

for (int j = 0; j < zRes; j++)

{

float zPos = ((float)j / (zRes - 1) - 0.5f) \* size;

verts[i + j \* zRes] = new Vector3(xPos, 0, zPos);

}

}

Using this method I could change the plane’s size and resolution on the fly.

Since I had the mesh’s vertices, I could store them as an array of Vector3s (saved as originalHeight[i]) and change, for the most part, their y values to match those of a sine wave with the use of Mathf.Sin:

At this point, the sine wave worked perfectly, and the user could change any of the effectors, such as the sine wave’s amplitude and frequency, to change the mesh in real-time.

originalHeight[] = GetComponent<MeshGen>().verts;

Vector3 vert = originalHeight[i];

//MOVE VERTS IN SINE WAVE

vert.y += Mathf.Sin(Time.time \* speed +

((originalHeight[i].x \* moveAmount.x) +

(originalHeight[i].z \* moveAmount.z)) \* scale) \* height;

//ADDING PERLIN NOISE FOR EFFECT

vert.y += Mathf.PerlinNoise(originalHeight[i].x, originalHeight[i].y + Mathf.Sin(Time.time \* 0.1f)) \* noiseScale;

vertices[i] = vert;

I also added a simple Perlin Noise effect in order to stop the sine waves from being too perfect. This effect can be disabled simply by passing in a 0 value through the noiseScale.

In order to give the tool more of a visual flair, I wanted to grab the edges of the mesh and bring them downwards, giving the effect of a waterfall or a cube of water suspended in space. My first attempt included checking each vertice and determining how many edges were attached to it, as the outer vertices would have less edges attached to them compared to the central vertices, but in the end I settled on a much simpler approach. The following code shows how I grab the left and bottom vertices of the mesh:

if ((i % (xRes)) == 0) //left edge

{

vertices[i] = new Vector3(vertices[i + 1].x, -500, vertices[i + 1].z);

}

else if (i < xRes) // bottom row

{

vertices[i] = new Vector3(vertices[xRes].x, -500, vertices[xRes].z);

}

The rest of the code deals mostly with the UI’s sliders and buttons, activating the wireframe camera and reading the numerous text files.

In order to make the scene more visually interesting, I constructed a simple lighthouse and deformed ‘sand’ plane in Blender3D.

How to Use

The user can change any of the sliders’ values on the left at any time, and the mesh will update. Since the mesh’s size and resolution are determined when the mesh is actually created (at the start of the scene), it can’t be updated in real time – the user has to reset the scene by pressing the ‘Generate’ button next to the ‘Wireframe’ checkbox. I’ve ensured that every other value is saved and crosses over to the new scene, so no work is lost.

Every value other than the mesh’s size and resolution can be changed in real-time. Simply move the corresponding slider and the mesh will update accordingly.

To see each vertex in action, the user can see the mesh in ‘Wireframe’ mode – simply check the ‘Wireframe’ box, or **use the ‘W’ key** to activate it. The active camera is changed to one that only renders the wave’s (and test balls) vertices.

The user can press **the ‘Z’ key** to instantiate a ball in the middle of the level, which can collide with the wave. This serves no real purpose, other than to show that the mesh’s collider updates in real-time.

Text Files

Unity3D has a unique class known as TextAssets, which allows raw text files located in the program’s project folders and to be read. In this case, the 8 buttons on the right load their respective text file and reset the scene. An example text file is laid out as such:

* 50 //MESH SIZE
* 50 //RESOLUTION
* 0.5 //WAVE HEIGHT
* 1 //WAVE SCALE
* 1 //WAVE SPEED
* 0 //NOISE SCALE
* 1 //X MULTIPLIER
* 1 //Z MULTIPLIER

The X and Z multipliers define the direction and frequency of the waves in the respective direction. Most of the prefabs have them both set to 1, making the wave move diagonally. Changing these values can give the wave a skewed appearance, or, if they’re both 0, a completely flat wave moving only by the Wave Height value.

The ‘Save’ and ‘Load’ buttons present on the lower half of the right side allow the user to save and load a custom wave at any time. I used the StreamWriter class in order to write directly to the text file, as such:

string path = "Assets/TextFiles/User1.txt";

StreamWriter writer = new StreamWriter(path,false);

for (int i = 0; i < sliders.Length; i++)

{

writer.WriteLine(sliders[i].value);

}

writer.Close();

AssetDatabase.ImportAsset(path);

Known Bugs

If the player loads a new prefab when the ‘Settings’ tab has been disabled, the scene will reload, but the wave will not have changed – I believe this is because the sliders values determine the layout of the new wave, and if they don’t ‘exist’, nothing changes.

References

Music:  
Vreeland, Richard. *Beacon*. Staten Island: N.p., 2012. online.

Websites:

Technologies, Unity. "Unity - Scripting API: GL.Wireframe". *Docs.unity3d.com*. N.p., 2017. Web. 22 Mar. 2017.

"Sinusoid". Encyclopedia of Mathematics. Springer. Retrieved December 8, 2013.

“Streamwriter Dotnetperls". *Dotnetperls.com*. N.p., 2017. Web. 26 Mar. 2017.Bull, John W.

Books:

*Numerical Analysis And Modelling In Geomechanics*. 1st ed. London: Spon, 2003. Print.

Bochner, S.; Chandrasekharan, K. (1949), Fourier Transforms, Princeton University Press.