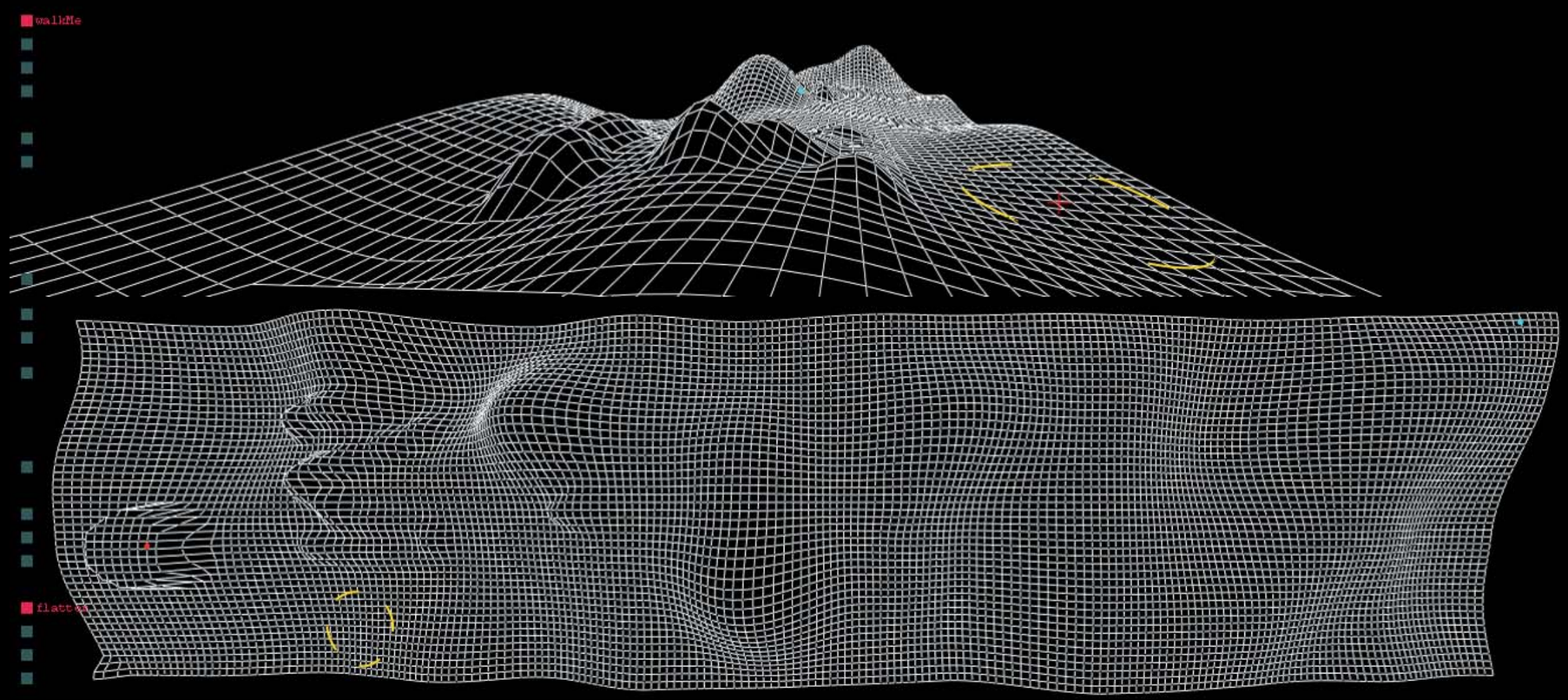


IMMERSIVE MODELING, the potential of video game technology as a design tool for landscape architecture

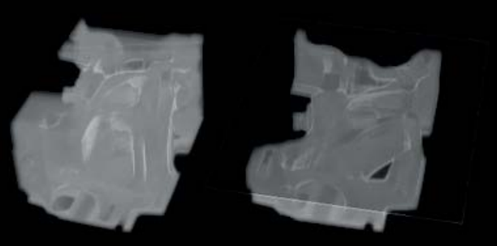
3D modeling software is nowadays the standard product architects and landscape architects use to design and communicate their envisioned designs to non-experts in the field. However, these platforms provide the designer with a highly abstracted environment to work with spatial geometry, usually from unnatural points of view to the human scale such as bird's eye, thus warping the designer's perception and understanding of their creation. A new approach to this design paradigm is rising from bringing video game technology, the so called game engines, into the production of 3D navigable environments for the purpose of conveying architectural experiences.

These experimental techniques are opening a vast space for enhanced design opportunities, when seen beyond mere representational tools and fully understood as a new modeling environment.

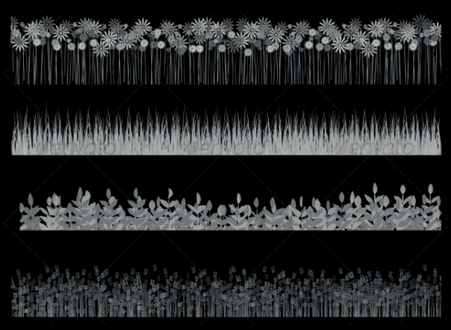
The aim of this project is to create a prototype of a topography sculpting platform, with special stress on the human scale and nature's logics.
The ambition is to impart the designer with an understanding and experience of being in the landscape that they are designing.

The rules for this platform will be:
-User interaction with the environment will strictly be bound to the first person view, fostering the understanding of the environment through the human scale.
-User movements will also be bound to real scale velocities, such as walking, running or driving, further enhancing a scaleful dynamic relation to the topography.
-Topography sculpting tools will be related to nature, and they will come in the form of water, rain, erosion, rifts, etc.
-Collaboration will be explored through the interaction of several agents concurrently in the same model, and a hierarchy of edition privileges.
-Geometry will be imported and exported for communication with other platforms.

Erosion / Deposition

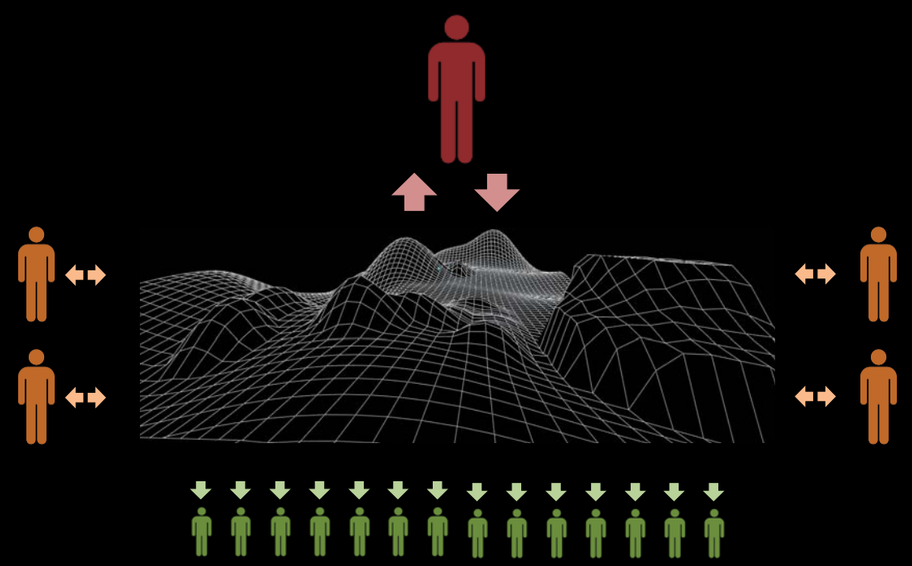


Grow / Harvest



Rain / Wind

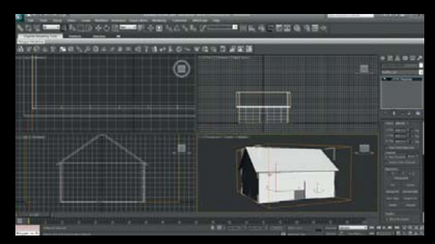
user hierarchy



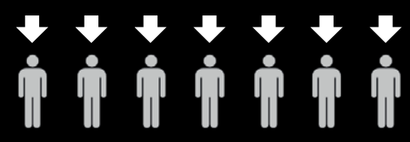
Techniques for Immersive Modeling

This would be an interface that would allow for less intrusive or disruptive interventions to a given landscape model. This is in line with Native American philosophies of minimal intervention. Consequently, as opposed to conventional modelling commands (extrude, loft, trim, etc.) which are scaleless and devoid of physical constraints, this platform instead provides modelling commands that emulate natural processes. This should allow for a more visceral and meaningful relationship between the designer and the landscape of interest.

An additional focus will be placed on modes of representation. As opposed to commercial model visualization packages like Lumion, this platform is more interested in abstraction over photorealism. It will provide a series of preset view modes intended to evoke a more qualitative understanding of the environment. Additionally the platform will support anaglyphic stereoscopy for enhanced spatial perception.

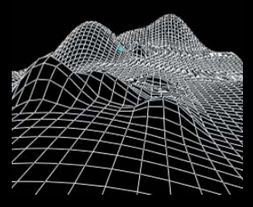


immersive modeling platform



References: Minecraft

References: From Dust



IMMERSIVE MODELING

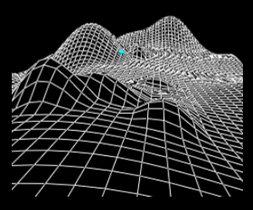
JOSE LUIS GARCÍA DEL CASTILLO / DDES'17
NATHAN MELENBRINK / MDESS'16

IMMERSIVE LANDSCAPE
representation through gaming technology

VIS - 0224900 - ERIC de BROCHE des COMBES / CHAD OPPENHEIM - FALL 2014

```
var heightMap : Texture2D;
//var heightMap : MovieTexture;
var material : Material;
var size = Vector3(200, 30, 200);

function Start ()
{
    GenerateHeightmap();
}

function Update ()
{
    var mesh : Mesh = GetComponent(MeshFilter).mesh;
    GetComponent(MeshCollider).mesh = GetComponent(MeshFilter).sharedMesh;
    transform.GetComponent(MeshCollider).sharedMesh = mesh;
}

function GenerateHeightmap ()
{
    // Create the game object containing the renderer
    gameObject.AddComponent(MeshFilter);
    gameObject.AddComponent(MeshRenderer);
    gameObject.AddComponent(MeshCollider);

    GetComponent(MeshCollider).mesh = null;
    GetComponent(MeshCollider).mesh = GetComponent(MeshFilter).sharedMesh;

    if (material)
        renderer.material = material;
    else
        renderer.material.color = Color.white;

    // this line of code will make the Movie Texture begin playing
    //renderer.material.mainTexture.Play();
    //renderer.material.mainTexture.loop = true;

    // Retrieve a mesh instance
    var mesh : Mesh = GetComponent(MeshFilter).mesh;
    var width : int = Mathf.Min(heightMap.width, 25);
    var height : int = Mathf.Min(heightMap.height, 25);
    var y = 0;
    var x = 0;

    // Build vertices and UVs
    var vertices = new Vector3[height * width];
    var uv = new Vector2[height * width];
    var tangents = new Vector4[height * width];

    var uvScale = Vector2 (1.0 / (width - 1), 1.0 / (height - 1));
    var sizeScale = Vector3 (size.x / (width - 1), size.y, size.z / (height - 1));

    for (y=0;y<height;y++)
    {
        for (x=0;x<width;x++)
        {
            var pixelHeight = heightMap.GetPixel(x, y).grayscale;
            var vertex = Vector3 (x, pixelHeight, y);
            vertices[y*width + x] = Vector3.Scale(sizeScale, vertex);
            uv[y*width + x] = Vector2.Scale(Vector2 (x, y), uvScale);

            // Calculate tangent vector: a vector that goes from previous vertex
            // to next along X direction. We need tangents if we intend to
            // use bumpmap shaders on the mesh.
            var vertexL = Vector3( x-1, heightMap.GetPixel(x-1, y).grayscale, y );
            var vertexR = Vector3( x+1, heightMap.GetPixel(x+1, y).grayscale, y );
            var tan = Vector3.Scale( sizeScale, vertexR - vertexL ).normalized;
            tangents[y*width + x] = Vector4( tan.x, tan.y, tan.z, -1.0 );
        }
    }

    // Assign them to the mesh
    mesh.vertices = vertices;
    mesh.uv = uv;

    // Build triangle indices: 3 indices into vertex array for each triangle
    var triangles = new int[(height - 1) * (width - 1) * 6];
    var index = 0;
    for (y=0;y<height-1;y++)
    {
        for (x=0;x<width-1;x++)
        {
            // For each grid cell output two triangles
            triangles[index++] = (y     * width) + x;
            triangles[index++] = ((y+1) * width) + x;
            triangles[index++] = (y     * width) + x + 1;

            triangles[index++] = ((y+1) * width) + x;
            triangles[index++] = ((y+1) * width) + x + 1;
            triangles[index++] = (y     * width) + x + 1;
        }
    }
    // And assign them to the mesh
    mesh.triangles = triangles;

    // Auto-calculate vertex normals from the mesh
    mesh.RecalculateNormals();

    // Assign tangents after recalculating normals
    mesh.tangents = tangents;

    //transform.gameObject.AddComponent(MeshCollider);
    transform.GetComponent(MeshCollider).sharedMesh = mesh;
}
```

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using Newtonsoft.Json;

public class MeshBuilder : MonoBehaviour {
    private System.IO.FileSystemWatcher m_Watcher;
    private GameObject[] grasshopperObjects;
    public string folder;
    public string filename;
    private string pathToGHObjects;
    public Texture mat1;
    public Vector2 scale;
    public Vector2 offset;
    public Color mesh_color;
    public Shader mesh_shader;

    // Use this for initialization
    void Start () {
        print ("Reticulating Splines");
        m_Watcher = new System.IO.FileSystemWatcher();
        m_Watcher.Path = folder;
        m_Watcher.Filter = filename;
        m_Watcher.NotifyFilter = System.IO.NotifyFilters.LastAccess | System.IO.NotifyFi
        m_Watcher.Changed += new System.IO.FileSystemEventHandler(OnChanged);
        m_Watcher.EnableRaisingEvents = true;

        meshHasChanged = true;
        pathToGHObjects = folder + filename;

    }

    void OnChanged(object sender, System.IO.FileSystemEventArgs e) {
        meshHasChanged = true;
        pathToGHObjects = e.FullPath;
    }

    void UpdateGHObjects ()
    {
        //print ("File has changed!");

        string json_string = System.IO.File.ReadAllText(pathToGHObjects);


        grasshopperObjects = new GameObject[json_dict.Count];

        for (int i = 0; i < json_dict.Count; i++)
        {
            Dictionary<string, List<float[]>> ghObject = json_dict[i];

            GameObject newGameObject = new GameObject();
            newGameObject.AddComponent<MeshFilter>();
            newGameObject.AddComponent<MeshRenderer>();
            newGameObject.AddComponent<MeshCollider>();

            newGameObject.GetComponent<MeshRenderer>().materials[0].mainTexture =mat1;
            newGameObject.GetComponent<MeshRenderer>().materials[0].mainTextureScale= sc
            newGameObject.GetComponent<MeshRenderer>().materials[0].mainTextureOffset= c
            newGameObject.GetComponent<MeshRenderer>().materials[0].color= mesh_color;
            newGameObject.GetComponent<MeshRenderer>().materials[0].shader = mesh_shader

            Mesh newMesh = new Mesh();

            Vector3[] newVertices;
            int[] newTriangles;
            Vector2[] newUVs;

            newVertices = new Vector3[ghObject["vertices"].Count];
            newTriangles = new int[ghObject["faces"].Count*3];
            newUVs = new Vector2[ghObject["vertices"].Count];

            for (int j = 0; j < ghObject["vertices"].Count; j++)
            {
                newVertices[j] = new Vector3(ghObject["vertices"][j][0], ghObject["verti
                newUVs[j] = new Vector2(ghObject["vertices"][j][0], ghObject["vertices"]
            }

            int counter = 0;

            for (int j = 0; j < ghObject["faces"].Count; j++){
                for ( int k = 0; k < 3; k++){
                    newTriangles[counter] = (int)ghObject["faces"][j][k];
                    counter++;
                }
            }

            newMesh.vertices = newVertices;
            newMesh.triangles = newTriangles;
            newMesh.uv = newUVs;
            newMesh.RecalculateNormals();
            newGameObject.GetComponent<MeshFilter>().mesh.Clear();
            newGameObject.GetComponent<MeshFilter>().mesh = newMesh;
            //newGameObject.GetComponent<MeshCollider>().sharedMesh.Clear();
            newGameObject.GetComponent<MeshCollider>().sharedMesh = newMesh;

            grasshopperObjects[i] = newGameObject;

        }

        meshHasChanged = false;
    }

    // Update is called once per frame
    void Update () {

        if (meshHasChanged == true)
        {
            UpdateGHObjects();
        }

    }
}
```



IMMERSIVE MODELING

JOSE LUIS GARCÍA DEL CASTILLO / DDES'17
NATHAN MELENBRINK / MDESS'16

IMMERSIVE LANDSCAPE
representation through gaming technology

VIS - 0224900 - ERIC de BROCHE des COMBES / CHAD OPPENHEIM - FALL 2014