Visual Computing I:

Interactive Computer Graphics and Vision

Scenegraphs/Game Engines

Stefanie Zollmann and Tobias Langlotz

Recap

WebGL

- Inside an HTML <canvas> element.
- Development is likely to involve HTML, CSS, JavaScript, GLSL, in addition to WebGL's OpenGL-style naming.

THREE.JS

- First released April 2010
- 3D Javascript Library
- Under MIT license
- Uses WebGL for rendering (previously also CanvasRenderer, SVGRenderer)
- Runs in all browsers that support Web
- Website: threejs.org
- Library is in single javascript file

Interactive: Object Picking

- Raycaster computes the intersection of a ray and a set of scene objects
- Raycaster used for mouse picking
- Given a mouse coordinate computes which objects in the 3d space are hit
- Returns an array of intersected objects

```
var raycaster = new THREE.Raycaster();
raycaster.setFromCamera( mouse, camera );
var intersects = raycaster.intersectObjects( scene.children );
```

Other webGL frameworks

- Cesium
  - Open-source WebGL framework for rendering globes and maps
  - Focus on visualising dynamic data
  - Support of labels, billboards
  - Sandbox

Other webGL frameworks

- Babylon.js:
  - JavaScript framework for building 3D games with HTML 5 and WebGL
  - Focus on games
  - Features like physics engine, shadows, assets management

https://www.babylonjs.com/Demos/Flat2009/

WEBXR

Supported devices

- ARCore-compatible devices
- Google Daydream
- HTC Vive
- Magic Leap One
- Microsoft Hololens
- Oculus Rift
- Samsung Gear VR
- Windows Mixed Reality headsets
- Apple Vision Pro (some adjustments needed)

	Headset Devices	Handheld Device e.g. Phone
VR	VR Devices, previously handled by WebVR	Magic Window Behaviour
AR	Mixed Reality Headsets	Phone AR

https://github.com/immersive-web/webxr

https://immersive-web.github.io/webxr/

TODAY

SCENE GRAPH

OPTIMISATIONS

GAME ENGINES

Why Scene Graphs?

- OpenGL, WebGL
  - Designed as a state machine
  - Only aware of primitives (e.g. triangles)
  - No knowledge about complete scene

Problem

```
void drawBike(){

    drawFrame();
    drawFrontWheel();
    drawRearWheel();

}
```

- Spatial relationship not stored
- How to move parts at the same time?
- Identical wheels are drawn twice

Why Scene Graphs?

- Scene Graphs
  - Store objects in a scene in a graph
  - Group related objects that share some properties together
  - Can optimise the way data is passed to the graphics hardware

Scene Graph

Root
Group
Group
Transform
Geometry
Material
Geometry
Geometry

Scene Graph

- Data structure
- Represented by a tree
  - Directed acyclic graph
  - Root on top
  - Leaves representing geometries
  - Objects are placed relative to each other
- Represents logical and spatial relationship between objects in a scene

Advantages

- Reuse geometry
- Reuse materials
- Optimisation based on view settings
- Optimisation based on occlusions
- Optimisation based on distance

https://docs.unity3d.com/560/Documentation/Manual/OcclusionCulling.html

Scene Graph

- Advantages:
  - Performance
  - Productivity
  - Portability
  - Scalability

https://docs.unity3d.com/560/Documentation/Manual/Materials.html

Scene Graph: Duplicate Objects

- Duplicated objects in our scene should share the same nodes in the scene graph
- Note that we can use nodes to hold matrices, as well as geometry
- The transformations in a matrix node will apply to all children of that node

Scene Graph: Subcomponents

- Further geometric decomposition:
  - Wheel is itself made up of subcomponents

Scene Graph: Transformations

- Object has a local transformation relative to its parent
- If bike root is transformed the frame and all child nodes would be transformed as well
- Root of the graph corresponds to “world coordinates”
- Each geometry is defined in local coordinates

World Coordinate System

- Everything is positioned relative to the world coordinates
- Model matrix: Convert points in an object's local coordinate system into world coordinate system
- View matrix: Convert world coordinates to viewing coordinates

Local Coordinate System

- An object's local transformation describes the mapping between the object's local coordinate system to the parent's local coordinate system
- E.g. the left spoke's transformation describes the mapping between the wheel and the spoke

Local Coordinate System

- Each part of the robot is modelled in its own local coordinate system
- If the transformation of a part is changed it affects all its child nodes

Compared to our initial problem

```
void drawBike(){

    drawFrame();
    drawFrontWheel();
    drawRearWheel();

}
```

- Spatial relationship not stored
- How to move parts at the same time?
- Identical wheels are drawn twice

Scene Graph Nodes

- Different types of nodes
  - Group Nodes
  - Transformation Nodes
  - Light Nodes
  - Geometry nodes (leaf nodes)
    - Polygons, lines, points
    - Material
  - Switch nodes
  - LOD nodes

Traversal

Rendering consists of traversing the graph:

- Geometry nodes have to be rendered (primitives, meshes),
- Transform node modifies transformation matrices
- State node change a render state (colour, material, shader)

Performance Optimisation

Level of Detail

- Rendering Optimisation
- Change level of detail depending on distance to the camera

Level of Detail

Image					
Vertices	~5500	~2880	~1580	~670	140
Notes	Maximum detail, for closeups.				Minimum detail, very far objects.

http://alphapixel.com/wp-content/uploads/2015/04/LOD-Level-of-detail-in-OpenSceneGraph-OSG.pdf

- Rendering Optimisation
- Change level of detail depending on distance to the camera

LOD Node

- Node allows switching between children depending on distance from eye point
- Objects further away from the eye point are rendered at a lower level of detail
- Each LOD child has a corresponding valid range consisting of a minimum and maximum distance:
  - Given a distance to the viewer (d), LOD displays a child if min <= d < max
- Usage like group node: lod.addChild(detailedNode);
- Set the visible range from the viewer to the object:
  - lod.setRange(childNumber, near, far);

LOD NODES

```
LOD lod = new LOD();

lod.addChild(highDetail);
lod.setRange(0, 0, 10);

lod.addChild(medDetail);
lod.setRange(1, 10, 100);

lod.addChild(lowDetail);
lod.setRange(2, 100, 1000);

lod.addChild(noDetails);
lod.setRange(2,1000,10000);
```

Billboards

Culling

- Scene graph structure supports efficient culling
  - Bounding volume is calculated at each node in the scene graph
  - Includes geometry of node and child nodes
  - When traversing the scene graph, bounding volumes are evaluated against view frustum
  - If bounding volume does not intersect with view frustum -> culled
  - Culled nodes and their children are not sent to the graphic card

https://docs.unity3d.com/460/Documentation/Manual/OcclusionCulling.html

Scene Graph Implementations

- OpenSceneGraph
- VulkanSceneGraph (VSG)
- OpenInventor
- OpenSG (http://www.opensg.org)
- COIN3D (API compatible with Open Inventor 2.1)

OpenSceneGraph

http://trac.openscenegraph.org/projects/osg/wiki/Screenshots/WSPSweden

OpenSceneGraph

The Foundation of the Hellenic World: http://trac.openscenegraph.org/projects/osg/wiki/Screenshots/FHW

OpenSceneGraph

https://live.osgeo.org/_images/osgearth3.jpg

OpenSceneGraph

By ALJI Mohamed - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=18209753

Geodes - Geometry Nodes

```
osg:: Vec3 vcen(1.0, 1.0, 1.0);

 osg::Cone* cone = new osg::Cone(vcen, 2.0, 10.0);
 // Create a drawable object based on the cone

 osg:: ShapeDrawable *drawable = new osg::ShapeDrawable(cone);

 drawable->setColor(osg::Vec4(1.0f, 0.0f, 0.0f, 1.0f));
 // create a new geode

 osg:: Geode* geode = new osg::Geode();
 geode->addDrawable(drawable);

  osg::Group * node = new osg::Group();
 node->addChild(geode);
```

Group Nodes

```
float r = node->getBound().radius(); // diameter
osg::Group * root = new osg::Group();
osg::Group * group = new osg::Group();

// Create four matrices for translated instances of the cow
osg::MatrixTransform * transform0  = new osg::MatrixTransform( );
transform0->setMatrix( osg::Matrix::translate( 0,0,r ) );

osg::MatrixTransform * transform1  = new osg::MatrixTransform( );
transform1->setMatrix( osg::Matrix::translate( 0,0,0 ) );

osg::MatrixTransform * transform2  = new osg::MatrixTransform( );
transform2->setMatrix( osg::Matrix::translate( -r,0,-r ) );

osg::MatrixTransform * transform3  = new osg::MatrixTransform( );
transform3->setMatrix( osg::Matrix::translate(  r,0,-r ) );

root->addChild( transform0 );
root->addChild( group );
group->addChild( transform1 );
group->addChild( transform2 );
group->addChild( transform3 );

transform0->addChild( node );
transform1->addChild( node );
transform2->addChild( node );
transform3->addChild( node );
```

Additional Libraries

OSG NODEKITS

- osgText - anti-aliased text
- osgFX - special effects framework
- osgShadow - shadow framework
- osgManipulator - 3d interactive controls
- osgTerrain - terrain rendering
- osgAnimation - character and rigid body animation
- osgVolume - volume rendering
- Etc.

VulkanSceneGraph

Core Features

- Vulkan-first architecture: Explicit command buffers, pipelines, and synchronization.
- Scene graph abstraction: Nodes, visitors, and traversal patterns similar to OSG.
- Efficient multi-threading: Parallel command graph generation.
- PBR-ready rendering: Materials, lights, and shaders aligned with modern pipelines.
- Interoperability: Support for glTF, common image formats, and custom data sources.

3D Geospatial Application Engine (Vulkan / C++17 / VSG) https://github.com/pelicanmapping/rocky.

Game Engines

Game Engines

- Provide:
  - Rendering
  - Scene graph
  - Animation
  - Physics Engine
- Visual UI for scene editing

Render/Game Engines

- Unity
- Unreal
- CryEngine
- Etc.

Basic Concepts

Project - All the elements that makes up the game, including models, assets, scripts, scenes, etc…

Scenes - Collection of game objects that constitute the world that the player sees at any time.

Packages: Aggregation of game objects and their associated metadata

Basic Concepts

Prefabs: A prefab is a template for grouping various assets under a single header.

- Prefabs are used for creating multiple instances of a common object.
- For example, you may have a large number of copies of a single element (e.g., street lights, trees)
- Prefabs can be instantiated during runtime

Overview

Object hierarchy

Assets

Game object inspector

Scene View

Creating a Geometry

Camera

- Not to be confused: scene camera vs editor camera
- Default is “Main Camera tagged as “MainCamera” in inspector for accessing the camera from a script
- “Camera Preview” shows a preview of the camera/ what camera can see
- Preview shows preview of play mode

UNITY Scene Graphs

- Collection of nodes in a graph or tree structure
- Specific to Unity: tree nodes have only a single parent but may have many children
- Operations (e.g. transformations) applied to a parent are applied to all its child nodes

Game Objects

- The game objects are all the “things” that constitute your scene:
  - Light sources
  - Audio sources
  - Cameras
  - Gameplay Logic
  - User Interface

- Everything is a game object

Game Objects

- Have a transform component consisting of position/rotation/scale
- Components give game object a behaviour
- Examples:
  - Mesh Filter
  - Mesh Renderer
  - Rigidbody
  - Colliders
  - VideoPlayer
  - Custom: Scripts that inherit from MonoBehaviour

Scripts

- Public variables will be accessible in the Inspector
- Components can also be modified by the inspector

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

public class MyBehaviourScript : MonoBehaviour
{
    // We can acess this via the inspector
    public float rotateFactor = 0.5f;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
      float amountToRotate = rotateFactor * Time.deltaTime
      Vector3 axis = new Vector3(0,1,0);
      this.transform.Rotate(axis, amountToRotate);
    }
}
```

Shaders

- Part of graphics pipeline
- Built-in Unity shaders
  - Standard shader support physically based shading (PBS)
  - Combines different shader types (such as Diffuse, Specular, Bumped Specular, Reflective) into a single shader
  - Uses the Disney model for diffuse component, GGX model for specular, with Smith Joint GGX visibility term and Schlick Fresnel approximation

https://docs.unity3d.com/Manual/shader-StandardShader.html

Custom Shaders

- Custom shader
  - Shaders are written in Cg/HLSL and wrapped in ShaderLab
  - https://docs.unity3d.com/Manual/ShadersOverview.html
  - Passing parameters from the inspector

```
Shader "Custom/NewSurfaceShader"
{
    Properties
    {
        _Color ("Color", Color) = (1,1,1,1)
        _MainTex ("Albedo (RGB)", 2D) = "white" {}
        _Glossiness ("Smoothness", Range(0,1)) = 0.5
        _Metallic ("Metallic", Range(0,1)) = 0.0
    }
    SubShader
    {
        Tags { "RenderType"="Opaque" }
        LOD 200
        Cull Off

        CGPROGRAM
        // Physically based Standard lighting model, and enable shadows on all light types
        #pragma surface surf Standard fullforwardshadows

        // Use shader model 3.0 target, to get nicer looking lighting
        #pragma target 3.0

        sampler2D _MainTex;

        struct Input
        {
            float2 uv_MainTex;
        };
```

Summary

SCENE GRAPH

OPTIMISATIONS

GAME ENGINES

The end!