Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Graphical Objects and Scene Graphs 1

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

Objectives

- Introduce graphical objects
- Generalize the notion of objects to include lights, cameras, attributes
- Introduce scene graphs

Limitations of Immediate Mode Graphics

- When we define a geometric object in an application, upon execution of the code the object is passed through the pipeline
- It then disappeared from the graphical system
- To redraw the object, either changed or the same, we had to reexecute the code
- Display lists provided only a partial solution to this problem

Retained Mode Graphics

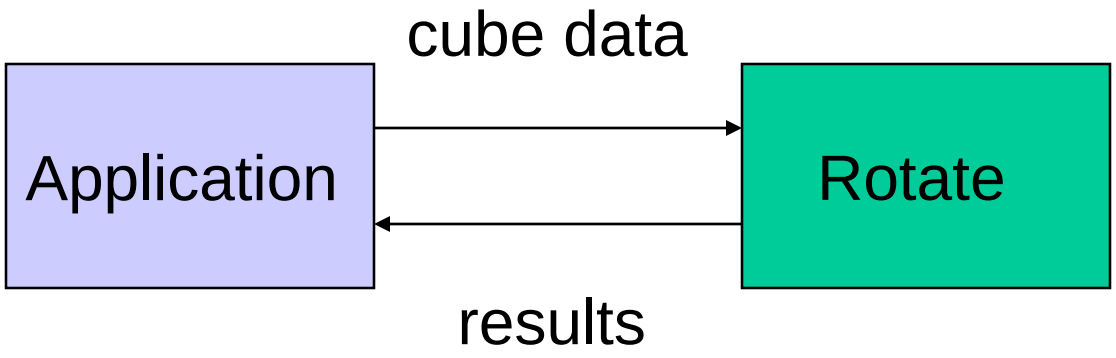
- Display lists were server side
- GPUs allowed data to be stored on GPU
- Essentially all immediate mode functions have been deprecated
- Nevertheless, OpenGL is a low level API

OpenGL and Objects

- OpenGL lacks an object orientation
- Consider, for example, a green sphere
  - We can model the sphere with polygons
  - Its color is determined by the OpenGL state and is not a property of the object
  - Loose linkage with vertex attributes
- Defies our notion of a physical object
- We can try to build better objects in code using object-oriented languages/techniques

Imperative Programming Model

- Example: rotate a cube



- The rotation function must know how the cube is represented
  - Vertex list
  - Edge list

Object-Oriented Programming Model

- In this model, the representation is stored with the object

- The application sends a *message* to the object
- The object contains functions (*methods*) which allow it to transform itself

C/C++/Java/JS

- Can try to use C structs to build objects
- C++/Java/JS provide better support
  - Use class construct
  - With C++ we can hide implementation using public, private, and protected members i
  - JS provides multiple methods for object

Cube Object

- Suppose that we want to create a simple cube object that we can scale, orient, position and set its color directly through code such as

```
var mycube = new Cube();
mycube.color[0]=1.0;
mycube.color[1]= mycube.color[2]=0.0;
mycube.matrix[0][0]=.....
```

Cube Object Functions

- We would also like to have functions that act on the cube such as
  - `mycube.translate(1.0, 0.0,0.0);`
  - `mycube.rotate(theta, 1.0, 0.0, 0.0);`
  - `setcolor(mycube, 1.0, 0.0, 0.0);`
- We also need a way of displaying the cube
  - `mycube.render();`

Building the Cube Object

```
var cube {
      var color[3];
      var matrix[4][4];

}
```

The Implementation

- Can use any implementation in the private part such as a vertex list
- The private part has access to public members and the implementation of class methods can use any implementation without making it visible
- Render method is tricky but it will invoke the standard OpenGL drawing functions

Other Objects

- Other objects have geometric aspects
  - Cameras
  - Light sources
- But we should be able to have nongeometric objects too
  - Materials
  - Colors
  - Transformations (matrices)

JS Objects

```
cube mycube;

material plastic;
mycube.setMaterial(plastic);

camera frontView;
frontView.position(x ,y, z);
```

JS Objects

- Can create much like Java or C++ objects
  - constructors
  - prototypes
  - methods
  - private methods and variables

```
var myCube = new Cube();
myCube.color = [1.0, 0.0, 0.0]’
myCube.instance = …….
```

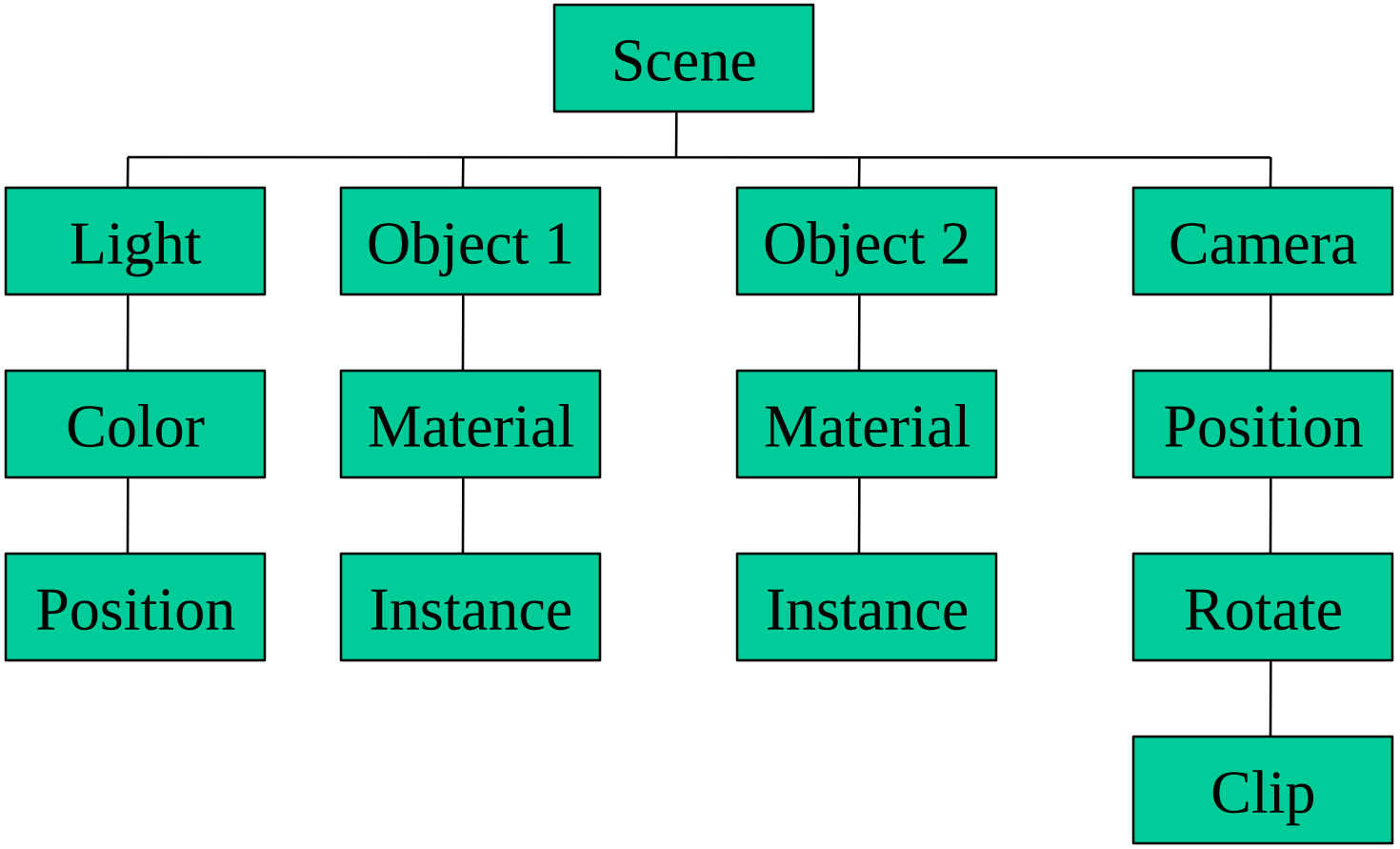
Light Object

```
var myLight = new Light();

// match Phong model

    myLight.type = 0; //directional
    myLight.position = ……;
    myLight.orientation = ……;
    myLight.specular = ……;
    myLight.diffuse = ……;
    myLight.ambient = ……;
}
```

Scene Descriptions

- If we recall figure model, we saw that
  - We could describe model either by tree or by equivalent code
  - We could write a generic traversal to display
- If we can represent all the elements of a scene (cameras, lights,materials, geometry) as JS objects, we should be able to show them in a tree
  - Render scene by traversing this tree

Scene Graph


Scene
Light
Object 1
Object 2
Camera
Color
Material
Material
Position
Position
Instance
Instance
Rotate
Clip

Traversal

```
myScene = new Scene();
myLight = new Light();
myLight.Color = ……;
…
myscene.Add(myLight);
object1 = new Object();
object1.color = …
myscene.add(object1);
…
…
myscene.render();
```

Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Graphical Objects and Scene Graphs 2

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

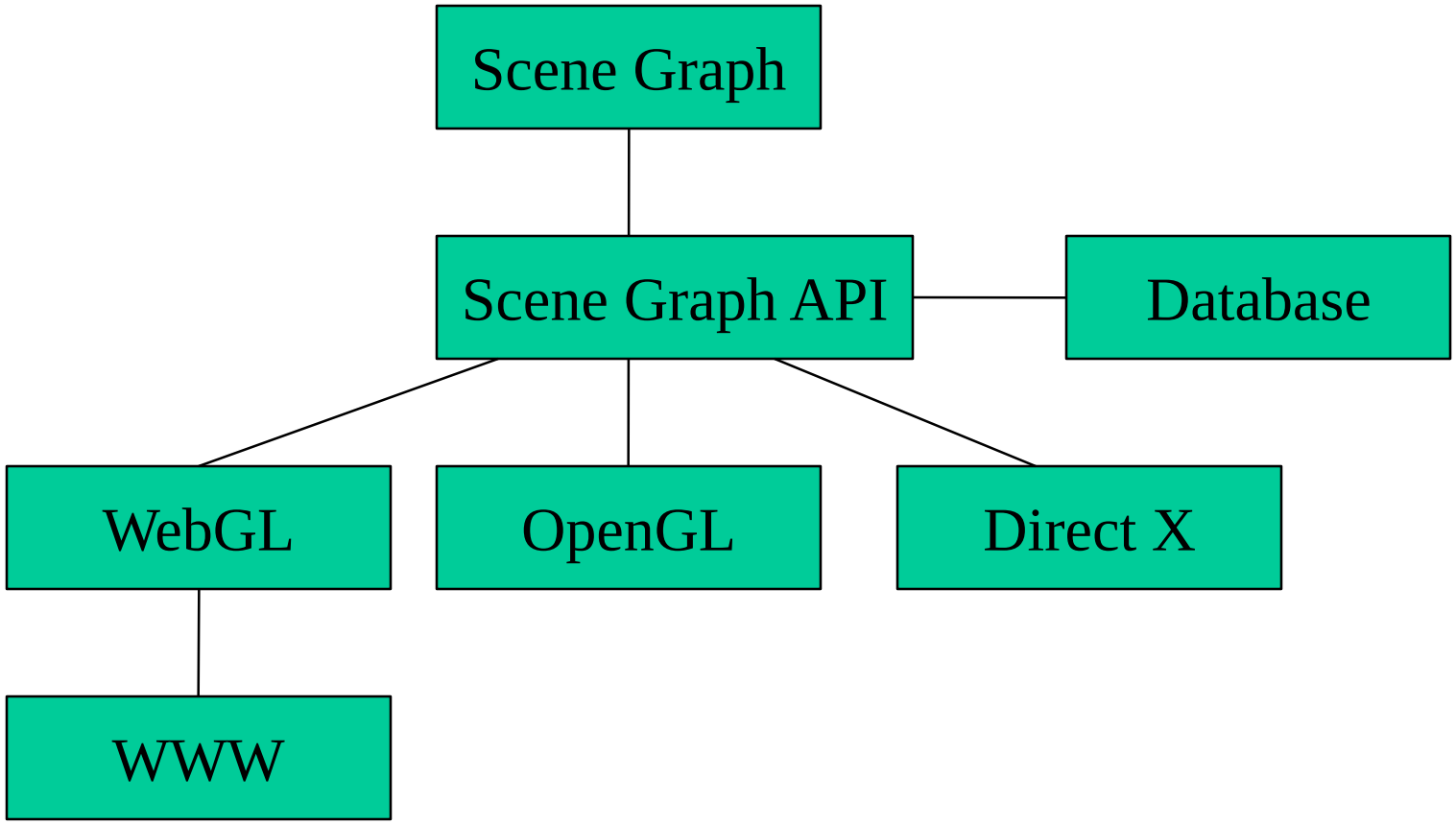
Objectives

- Look at some real scene graphs
- three.js (threejs.org)
- Scene graph rendering

Scene Graph History

- OpenGL development based largely on people who wanted to exploit hardware
  - real time graphics
  - animation and simulation
  - stand-alone applications
- CAD community needed to be able to share databases
  - real time not and photorealism not issues
  - need cross-platform capability
  - first attempt: PHIGS

Scene Graph Organization



Inventor and Java3D

- Inventor and Java3D provide a scene graph API
- Scene graphs can also be described by a file (text or binary)
  - Implementation independent way of transporting scenes
  - Supported by scene graph APIs
- However, primitives supported should match capabilities of graphics systems
  - Hence most scene graph APIs are built on top of OpenGL, WebGL or DirectX (for PCs)

VRML

- Want to have a scene graph that can be used over the World Wide Web
- Need links to other sites to support distributed data bases
- Virtual Reality Markup Language
  - Based on Inventor data base
  - Implemented with OpenGL

Open Scene Graph

- Supports very complex geometries by adding occulusion culling in first pass
- Supports translucently through a second pass that sorts the geometry
- First two passes yield a geometry list that is rendered by the pipeline in a third pass

three.js

- Popular scene graph built on top of WebGL
  - also supports other renderers
- See threejs.org
  - easy to download
  - many examples
- Also Eric Haines' Udacity course
- Major differences in approaches to computer graphics

three.js scene

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(75,
window.innerWidth/          window.innerHeight, 0.1, 1000);

 var renderer = new THREE.WebGLRenderer();
 renderer.setSize(window.innerWidth, window.innerHeight);
 document.body.appendChild(renderer.domElement);

 var geometry = new THREE.CubeGeometry(1,1,1);
 var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
 var cube = new THREE.Mesh(geometry, material);
 scene.add(cube);
 camera.position.z = 5;
```

three.js render loop

```
var render = function () {
 requestAnimationFrame(render);
 cube.rotation.x += 0.1;
 cube.rotation.y += 0.1;
 renderer.render(scene, camera);
 };
 render();
```

Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Rendering Overview

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- Examine what happens between the vertex shader and the fragment shader
- Introduce basic implementation strategies
- Clipping
- Rendering
  - lines
  - polygons
- Give a sample algorithm for each

Overview

- At end of the geometric pipeline, vertices have been assembled into primitives
- Must clip out primitives that are outside the view frustum
  - Algorithms based on representing primitives by lists of vertices
- Must find which pixels can be affected by each primitive
  - Fragment generation
  - Rasterization or scan conversion

The University of New Mexico

Required Tasks

- Clipping
- Rasterization or scan conversion
- Transformations
- Some tasks deferred until fragment processing
  - Hidden surface removal
  - Antialiasing

Rasterization Meta Algorithms

- Any rendering method process every object and must assign a color to every pixel
- Think of rendering algorithms as two loops
  - over objects
  - over pixels
- The order of these loops defines two strategies
  - image oriented
  - object oriented

Object Space Approach

- **For every object**, determine which pixels it covers and shade these pixels
  - Pipeline approach
  - Must keep track of depths for HSR
  - Cannot handle most global lighting calculations
  - Need entire framebuffer available at all times

Image Space Approach

- **For every pixel**, determine which object that projects on the pixel is closest to the viewer and compute the shade of this pixel
  - Ray tracing paradigm
  - Need all objects available
- Patch Renderers
  - Divide framebuffer into small patches
  - Determine which objects affect each patch
  - Used in limited power devices such as cell phones

Algorithm Experimentation

- Create a framebuffer object and use render-to-texture to create a virtual framebuffer into which you can write individual pixels