Overview of Computer Graphics

1-13-15
Outline

- What is Computer Graphics?
  - a historical perspective
- Draw a triangle using WebGL
Computer Graphics

- Computer graphics deals with all aspects of creating images with a computer
  - Hardware
  - Software
  - Applications
Example

- Where did this image come from?

- What hardware/software did we need to produce it?
Preliminary Answer

- **Application**: The object is an artist’s rendition of the sun for an animation to be shown in a domed environment (planetarium)

- **Software**: Maya for modeling and rendering but Maya is built on top of OpenGL

- **Hardware**: PC with graphics card for modeling and rendering
Basic Graphics System

Input devices

Image formed in frame buffer

Output device

- Computer graphics goes back to the earliest days of computing
  - Strip charts
  - Pen plotters
  - Simple displays using A/D converters to go from computer to calligraphic CRT
- Cost of refresh for CRT too high
  - Computers slow, expensive, unreliable
Direct View Storage Tube

- Created by Tektronix
  - Did not require constant refresh
  - Standard interface to computers
    - Allowed for standard software
    - Plot3D in Fortran
  - Relatively inexpensive
    - Opened door to use of computer graphics for CAD community
CRT

Can be used either as a line-drawing device (calligraphic) or to display contents of frame buffer (raster mode)
Shadow Mask CRT

- Wireframe graphics
  - Draw only lines
- Sketchpad
- Display Processors
- Storage tube

wireframe representation of sun object
Sketchpad

- Ivan Sutherland’s PhD thesis at MIT
  - Recognized the potential of man-machine interaction
  - Loop
    - Display something
    - User moves light pen
    - Computer generates new display
  - Sutherland also created many of the now common algorithms for computer graphics
Display Processor

- Rather than have the host computer try to refresh display use a special purpose computer called a display processor (DPU)

- Graphics stored in display list (display file) on display processor
- Host *compiles* display list and sends to DPU

- Raster Graphics
- Beginning of graphics standards
  - IFIPS
    - GKS: European effort
      - Becomes ISO 2D standard
    - Core: North American effort
      - 3D but fails to become ISO standard
- Workstations and PCs
Image produced as an array (the *raster*) of picture elements (*pixels*) in the *frame buffer*.
Raster Graphics

- Allows us to go from lines and wire frame images to filled polygons
Although we no longer make the distinction between workstations and PCs, historically they evolved from different roots

- Early workstations characterized by
  - Networked connection: client-server model
  - High-level of interactivity
- Early PCs included frame buffer as part of user memory
  - Easy to change contents and create images

Realism comes to computer graphics

smooth shading  environment mapping  bump mapping

- Special purpose hardware
  - Silicon Graphics geometry engine
    - VLSI implementation of graphics pipeline
- Industry-based standards
  - PHIGS
  - RenderMan
- Networked graphics: X Window System
- Human-Computer Interface (HCI)

- OpenGL API
- Completely computer-generated feature-length movies (Toy Story) are successful
- New hardware capabilities
  - Texture mapping
  - Blending
  - Accumulation, stencil buffers

- Photorealism
- Graphics cards for PCs dominate market
  - Nvidia, ATI
- Game boxes and game players determine direction of market
- Computer graphics routine in movie industry: Maya, Lightwave
- Programmable pipelines
- New display technologies
Generic Flat Panel Display

Vertical grid
Light emitting elements
Horizontal grid
Graphics is now ubiquitous
  Cell phones
  Embedded
• OpenGL ES and WebGL
• Alternate and Enhanced Reality
• 3D Movies and TV
The Programmer’s Interface

Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)
API Contents

- Functions that specify what we need to form an image
  - Objects
  - Viewer
  - Light Source(s)
  - Materials

- Other information
  - Input from devices such as mouse and keyboard
  - Capabilities of system
Object Specification

- Most APIs support a limited set of primitives including:
  - Points (0D object)
  - Line segments (1D objects)
  - Polygons (2D objects)
  - Some curves and surfaces
    - Quadrics
    - Parametric polynomials
- All are defined through locations in space or vertices
IFIPS (1973) formed two committees to come up with a standard graphics API
- Graphical Kernel System (GKS)
  - 2D but contained good workstation model
- Core
  - Both 2D and 3D
- GKS adopted as ISO and later ANSI standard (1980s)

GKS not easily extended to 3D (GKS-3D)
- Far behind hardware development
PHIGS and X

- Programmers Hierarchical Graphics System (PHIGS)
  - Arose from CAD community
  - Database model with retained graphics (structures)

- X Window System
  - DEC/MIT effort
  - Client-server architecture with graphics

- PEX combined the two
  - Not easy to use (all the defects of each)
Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982).

To access the system, application programmers used a library called GL.

With GL, it was relatively simple to program three dimensional interactive applications.
OpenGL

The success of GL lead to OpenGL (1992), a platform-independent API that was

- Easy to use
- Close enough to the hardware to get excellent performance
- Focus on rendering
- Omitted windowing and input to avoid window system dependencies
OpenGL Evolution

- Originally controlled by an Architectural Review Board (ARB)
  - Members included SGI, Microsoft, Nvidia, HP, 3DLabs, IBM, .......
  - Now Khronos Group
- Was relatively stable (through version 2.5)
  - Backward compatible
  - Evolution reflected new hardware capabilities
    - 3D texture mapping and texture objects
    - Vertex and fragment programs
- Allows platform specific features through extensions
Modern OpenGL

- A low-level graphics library (API) for 2D and 3D
- Descendant of GL (from SGI)
- First version in 1992; now: 4.5 in 2015
- Managed by Khronos Group (non-profit consortium)
- API is governed by Architecture Review Board (part of Khronos)

opengl.org
Modern OpenGL

- Performance is achieved by using GPU rather than CPU
- Control GPU through programs called shaders
- Application’s job is to send data to GPU
- GPU does all rendering
Example (old style)

```c
void display(void) {

    glBegin(GL_POLYGON);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, 0.0, 1.0);
    glEnd();
}
```

- type of object
- location of vertex
- end of object definition
GPU-based Example

- Put geometric data in an array
  ```cpp
  vec3 points[3];
  points[0] = vec3(0.0, 0.0, 0.0);
  points[1] = vec3(0.0, 1.0, 0.0);
  points[2] = vec3(0.0, 0.0, 1.0);
  ```
- Send array to GPU
- Tell GPU to render as triangle
Graphics Library (API)

- Intermediary between applications and graphics hardware

- Other popular APIs:
  - Direct3D (Microsoft)
  - OpenGL ES (embedded devices)
  - X3D (successor of VRML)
What About Direct X?

- Windows only
- Advantages
  - Better control of resources
  - Access to high level functionality
- Disadvantages
  - New versions not backward compatible
  - Windows only
- Recent advances in shaders are leading to convergence with OpenGL
Simplified Pipeline Model

Application ➔ GPU Data Flow ➔ Framebuffer

Vertices ➔ Vertices ➔ Fragments ➔ Pixels

- Vertex Processing
- Rasterizer
- Fragment Processing
- Vertex Shader
- Fragment Shader
Coding in WebGL

- Can run WebGL on any recent browser
  - Chrome
  - Firefox
  - Safari
  - IE
- Code written in JavaScript
- JS runs within browser
  - Use local resources
All WebGL programs must do the following:

- Set up canvas to render onto
- Generate data in application
- Create shader programs
- Create buffer objects and load data into them
- “Connect” data locations with shader variables
- Render
Simple Example, revisited

• Example: Draw a triangle
  Each application consists of (at least) two files
  HTML file and a JavaScript file

• HTML
  describes page
  includes utilities
  includes shaders

• JavaScript
  contains the graphics
<!DOCTYPE html>
<html>
<head>
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
void main(){
  gl_Position = vPosition;
}
</script>
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
void main(){
  gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
</script>
</head>
<body>
</body>
</html>
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="triangle.js"></script>
</head>
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
var gl;
var points;

window.onload = function init(){
    var canvas = document.getElementById( "gl-canvas" );
    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" );

    }

    // Three Vertices

    var vertices = [
        vec2( -1, -1 ),
        vec2( 0, 1 ),
        vec2( 1, -1 )
    ];
// Configure WebGL
gl.viewport( 0, 0, canvas.width, canvas.height );
gl clearColor( 1.0, 1.0, 1.0, 1.0 );

// Load shaders and initialize attribute buffers
var program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

// Load the data into the GPU
var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );
// Associate our shader variables with our data buffer

var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
render();

function render() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLES, 0, 3 );
}
ToDo for next class

- Create a github account (if you don’t have one)
- Download the source code (subdirectory CLASS) and utilities (subdirectory COMMON) from the Angel website
  - initShaders.js (initShaders2.js)
  - MV.js
  - webgl-utils.js
- Run the triangle.html file from the website
- Load the triangle.html and triangle.js files to your own computer, and run them locally
- Edit the files to change the color of the triangle, and to display two triangles

Experiment, have fun!
JavaScript Notes

• JavaScript (JS) is the language of the Web
  All browsers will execute JS code
  JavaScript is an interpreted object-oriented language

• References
  Flanagan, JavaScript: The Definitive Guide, O’Reilly
  Crockford, JavaScript, The Good Parts, O’Reilly
  Many Web tutorials
JS Notes

• Is JS slow?
  JS engines in browsers are getting much faster
  Not a key issues for graphics since once we get the data to the GPU it doesn’t matter how we got the data there

• JS is a (too) big language
  We don’t need to use it all
  Choose parts we want to use
  Don’t try to make your code look like C or Java
JS Notes

• Very few native types:
  numbers
  strings
  booleans

• Only one numerical type: 32 bit float
  var x = 1;
  var x = 1.0; // same
  potential issue in loops
  two operators for equality == and ===

• Dynamic typing
Scoping

• Different from other languages
• Function scope
  • variables are *hoisted* within a function
    can use a variable before it is declared
• Note functions are first class objects in JS
JS Arrays

- JS arrays are objects
  - inherit methods
  - `var a = [1, 2, 3];`
    - is not the same as in C++ or Java
  - `a.length  // 3`
  - `a.push(4);  // length now 4`
  - `a.pop();    // 4`
  - avoids use of many loops and indexing
  - Problem for WebGL which expects C-style arrays
Typed Arrays

JS has typed arrays that are like C arrays

```javascript
var a = new Float32Array(3)
var b = new Uint8Array(3)
```

Generally, we prefer to work with standard JS arrays and convert to typed arrays only when we need to send data to the GPU with the flatten function in MV.js
A Minimalist Approach

- We will use only core JS and HTML, no extras or variants.
- No additional packages.
  - CSS
  - JQuery
- Focus on graphics.
  - Examples may lack beauty.
- You are welcome to use other variants as long as I can run them from your URL.
Next topic: Image Formation

- Fundamental imaging notions
- Physical basis for image formation
  - Light
  - Color
  - Perception
- Synthetic camera model
- Other models