Computer Graphics Crash Course

CS 481/681 Computer Graphics Rendering

University of Alaska Fairbanks

Computer Graphics Crash Course

Overview

- Vectors
- Matrices
- Transformations
- WebGL Pipeline
- WebGL Shaders

Vectors

- $$\mathbf{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$
- Represents magnitude and direction
- *Homogeneous coordinates* adds a fourth w element
- Now we can multiply 4x4 matrices
- And we can project to 3D by dividing by the w component
- $$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \rightarrow \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{pmatrix}$$
- If we set $w = 1$, then we can represent position.
- If we set $w = 0$, then we only represent direction.

Data Layout

```
class Vector3 {
    constructor(x, y, z) {
        this.x = x || 0;
        this.y = y || 0;
```

```
        this.z = z || 0;
    }
}
```

Common Operations

- `let a = new Vector3()`
- `let b = Vector3.make(x, y, z)`
- `a.add(b)`
- `a.sub(b)`
- `a.negate()`
- `a.compMul(b)`
- `a.compDiv(b)`
- `a.scale(scalar)`

Vector Operations

- `a.dot(b);`
- `a.cross(b);`
- `a.length();`
- `a.norm();`
- `a.asArray();` returns `Float32Array`
- `Vector3.make(x, y, z)`
- `Vector3.makeUnit(x, y, z)`

Matrices

- $$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix}$$
- m_{ij} is ith row and jth column
- Represents orientation and origin
- $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ are axes and $\mathbf{o}$ is the origin of the system
- $$\mathbf{M} = \begin{pmatrix} \mathbf{u}_x & \mathbf{v}_x & \mathbf{w}_x & \mathbf{o}_x \\ \mathbf{u}_y & \mathbf{v}_y & \mathbf{w}_y & \mathbf{o}_y \\ \mathbf{u}_z & \mathbf{v}_z & \mathbf{w}_z & \mathbf{o}_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Data Layout

```
class Matrix4 {
    // data is column major
    // members are m{row}{col}
```

```
    constructor(
        m11, m21, m31, m41,
        m12, m22, m32, m42,
        m13, m23, m33, m43,
        m14, m24, m34, m44
    ) { // set members to Identity matrix}
}
```

Common Operations

- `let a = new Matrix4()`
- `let b = new Matrix4()`
- `a.add(b)`
- `a.sub(b)`
- `a.negate()`
- `a.compMul(b)`
- `a.compDiv(b)`
- `a.scale(scalar)`

Operations

- `M.determinant()` returns $\det \mathbf{A}$
- `M.inverse()` returns $\mathbf{M}^{-1}$
- `M.transpose()` returns $\mathbf{M}^T$
- `A.mult(B)` returns $\mathbf{AB}$
- `A.multMatrix(B)` computes $\mathbf{A} \leftarrow \mathbf{AB}$
- `A.loadMatrix()` computes $\mathbf{A} \leftarrow \mathbf{I}$

Construction and Array Conversion

- `let M = new Matrix4()` creates identity matrix
- `Matrix4.makeColMajor(m11, m21, m31, m41, ...)`
- `Matrix4.makeRowMajor(m11, m12, m13, m14, ...)`
- `M.asColMajorArray()` returns `[m11, m21, m31, m41, ...]`
- `M.asRowMajorArray()` returns `[m11, m12, m13, m14, ...]`

Interactions with Vector3

- Get a column vector: `let a = M.col3(1)`
- Get a row vector: `let b = M.row3(1)`
- Transform: `let b = M.transform3(a)` computes $\mathbf{M}(a_x\ a_y\ a_z\ 1)^T$

Transformations

Matrix4 Transformations

- `let m = new Matrix4()`
- `let I = Matrix4.makeIdentity()`
- `let T = Matrix4.makeTranslation(x, y, z)`
- `let S = Matrix4.makeScaling(x, y, z)`
- `let R = Matrix4.makeRotation(angleInDegrees, x, y, z)`
- `let P = Matrix4.makePerspectiveY(fieldOfViewY, aspectRatio, zNear, zFar)`
- `let C = Matrix4.makeLookAt(origin, center, up)`

WebGL Pipeline

- What are the essentials to rendering an object?
- Vertex Shader
- Fragment Shader
- Shader Program
- Array Buffer and Vertex Attrib Pointers
- Uniform Variables and Texture Maps
- Draw Call

WebGL Shaders

- Create/Compile a Vertex Shader
- Create/Compile a Fragment Shader
- Create a Shader Program
- Attach shaders and link
- WebGL 1.0 does not have
- Computer Shaders
- Geometry Shaders
- And so on ...

Graphics with the LibXOR Library

Using the LibXOR Library

1. Create a `div` with an id
2. Import the LibXOR javascript library
3. Import or embed your application
4. Latest version is at my GitHub LibXOR site
5. Ignore all the documentation for now, lots of things are in flux

Example 1 Hello, World WebGL

Example 2 Adding a shader and geometry

Example 3 Loading shaders and geometry

A Simple HTML5 Example

```
<!-- make a div as a container for the library -->
<div id='graphics'></div>
<!-- include LibXOR library -->
<script src="LibXOR.js"></script>
<script>/* Your code here */
```

Vertex shader

```
uniform mat4 ProjectionMatrix;
uniform mat4 CameraMatrix;
uniform mat4 WorldMatrix;

attribute vec3 aPosition;
attribute vec3 aNormal;
attribute vec3 aTexcoord;
attribute vec4 aColor;

// These MUST match the fragment shader
varying vec4 vPosition;
varying vec3 vNormal;
varying vec3 vTexcoord;
varying vec4 vColor;
```

Vertex shader

```
void main() {
    vNormal = uWorldMatrix * vec4(aPosition, 0.0);
    vColor = aColor;
    vTexcoord = aTexcoord;
    vPosition = uWorldMatrix * vec4(aPosition, 1.0);
    gl_Position = ProjectionMatrix * CameraMatrix * vPosition;
}
```

Fragment shader

```
uniform sampler2D map_kd;
uniform sampler2D map_ks;
uniform sampler2D map_normal;
uniform float map_kd_mix;
```

```
uniform float map_ks_mix;
uniform float map_normal_mix;
uniform vec3 kd;
uniform vec3 ks;

uniform vec3 sunDirTo;
uniform vec3 sunE0;
```

Fragment shader

```
// These MUST match the vertex shader
varying vec4 vPosition;
varying vec3 vNormal;
varying vec3 vTexcoord;
varying vec4 vColor;

void main() {
    // set to white
    gl_FragColor = vec4(1.0);
}
```

App class

```
class App {
    constructor() {
        // Set the id of the containing DIV
        this.xor = new LibXOR('graphics');
    }

    start() {
        this.mainloop();
    }
    // ...
}
```

Init Function

```
    init() {
        let xor = this.xor;
        let gl = xor.gl;

        // Initialize the graphics system
        this.xor.graphics.setVideoMode(576, 384);

        // create shader program
        let rc = this.xor.renderconfigs.load('basic', 'basic.vert', 'basic.frag');
```

```
        rc.depthTest = gl.LESS;
        rc.enableDepthTest = true;

        // or we can compile from strings
        rc = this.xor.renderconfigs.create('default');
        rc.compile(vshader, fshader); // variables declared elsewhere
```

Init function

```
        // create a mesh
        let rect = this.xor.meshes.create('rect');
        rect.begin(gl.TRIANGLE_FAN);
        rect.normal(0, 0, 1);
        rect.color(1, 1, 1, 1);
        rect.texcoord(0, 0, 0); rect.vertex(0, 0, 0);
        rect.texcoord(0, 1, 0); rect.vertex(0, 1, 0);
        rect.texcoord(1, 1, 0); rect.vertex(1, 1, 0);
        rect.texcoord(1, 0, 0); rect.vertex(1, 0, 0);

        // or load one
        this.xor.meshes.load('teapot', 'teapot.obj');
    }
```

Update and Render Functions

```
    update(timeInSeconds) { /* update state */ }

    render() {
        xor.graphics.clear();
        let projectionMatrix = Matrix4.makePerspective(45.0, xor.graphics.aspectRatio, 1.0,
        let cameraMatrix = Matrix4.makeOrbit(45.0, 45.0, 5.0);
        let rc = xor.renderconfigs.use('default');
        if (rc) {
            rc.uniformMatrix4f('ProjectionMatrix', projectionMatrix);
            rc.uniformMatrix4f('CameraMatrix', cameraMatrix);
            rc.uniformMatrix4f('WorldMatrix', Matrix4.makeRotation(xor.t1, 0, 1, 0));
            xor.meshes.render('teapot');
        }
        xor.renderconfigs.use(null);
    }
```

Mainloop function

```
    mainloop() {
        let self = this;
        window.requestAnimationFrame((t) => {
```

```
            xor.startFrame(t);
            self.update(xor.deltaTime);
            self.render();
            self.mainloop();
        })
    }
```

- And the actual app instantiation and starting...

```
let app = new App();
app.init();
app.start();
</script>
```

Hybrid Topics

Ray Tracing

- Watch YouTube Video
- Complete Activity Worksheet