Lab04. WebGL 3D- Drawing Cube

DoHoon Lee Ph.D

Visual & Biomedical Computing(VisBiC) Lab.
School of Computer Science & Engineering
Pusan National University

http://visbic.pusan.ac.kr/

WebGL Components

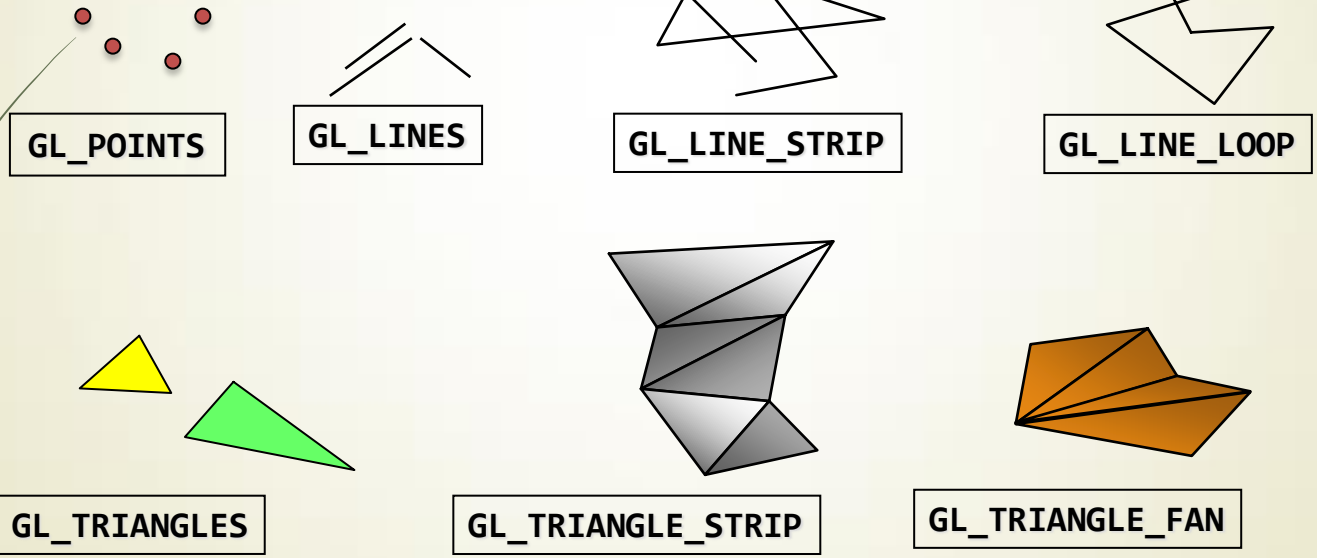
- Drawing buffer
  - Buffer를 읽고 쓸 수 있는 memory block, 임시 데이터를 저장하는 공간
- Primitive types
  - 특정 그래픽 언어에서 model을 만들 때 사용하는 graphic components
- Vertex data
  - 정점 버퍼에 보낼 정점에 대한 위치, 색상, 법선, 텍스처좌표와 같은 정점 특성(attributes)
- Vertex Buffer Object(VBO)
  - 정점의 특정 attribute에 대한 데이터를 보관하는 객체
- Attribute and uniform
  - Shader에 넘겨줄 수 있는 attribute 와 유니폼

Drawing buffers

- Color buffer
  - 색상 정보(red, green, blue)와 투명/불투명(알파값) 저장
- Depth buffer
  - Pixel의 깊이(심도, z값)에 대한 정보 저장
  - 3D를 2D 화면 공간에 매핑하다 보면 여러 점이 캔버스 상의 같은 (x,y) 위치에 투영될 수 있는데 이때 z값을 비교해서 가까운 한 점을 보관하고 rendering 함
- Stencial buffer
  - Rendering 하거나 rendering하지 않을 영역의 경계를 정하는데 사용
  - 이미지의 한 영역을 렌더링하지 않도록 표시하는 것을 해당 영역을 마스크한다고 함
  - 마스크한 영역을 포함한 전체 이미지를 스텐실이라 함
  - 스텐실버퍼는 심도버퍼오 함께 사용해 보이지 않는 영역을 렌더링하지 않음으로써 성능을 최적화하는데 활용

Primitive types

- WebGL : 7개의 primitive types
  - POINTS, LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN



Vertex data

- WebGL은 정점의 색상이나 위치를 직접 scene에 설정할 수 없음
- WebGL이 고정기능을 갖고 있지 않으며 프로그래밍 가능 세이더를 사용함
- 정점과 관련된 모든 데이터는 자바스크립트 API에서 GPU로 전달돼야 함
- WebGL에서는 위치, 색상, 법선, 텍스처 좌표 같은 정점 attribute를 보관하는 정점버퍼객체(VBO, Vertex Buffer Object)를 생성해야 함
- 그런 다음 정점 버퍼를 처리할 수 있는 세이더 프로그램으로 정점 버퍼를 보내야 함
- 고정기능 대신 shader를 사용한다는 점은 WebGL의 핵심적인 특징

Vertex Buffer Object(VBO)

- 각 VBO는 정점의 특정 attribute에 대한 데이터를 보관
- 이 attribute는 위치, 색상, 법선벡터, 텍스처 좌표 등임
  - 버퍼생성을 위한 WebGL API 호출 방식

```
var myBuffer = gl.createBuffer(); //WebGLBuffer생성하고 반환받은 객체를 저장함
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, myBuffer); //버퍼를 바인딩함
// void bindBuffer(Glenum target, WebGLBuffer buffer);
// target: gl.ARRAY_BUFFER or gl.ELEMENT_ARRAY_BUFFER
// gl.ARRAY_BUFFER : 위치 및 색상 같은 정점 attribute에 사용함
// gl.ELEMENT_ARRAY_BUFFER : 버퍼가 정점 index를 포함할 때 사용
gl.bufferData(gl.ARRAY_BUFFER, data, gl.STATIC_DRAW);
// void bufferData(Glenum target, ArrayBuffer data, Glenum usage);
// buffer를 바인딩하고 타입을 설정하고 데이터 저장
// usage : STATIC_DRAW, DYNAMIC_DRAW, STREAM_DRAW
// STATIC_DRAW : 한번 설정 여러 번 사용-변경되지 않음
// DYNAMIC_DRAW : 응용에서 데이터 여러 번 사용, 매번 재설정
// STREAM_DRAW : 데이터 변경안되며 응용에서 몇차례만 사용
```

Attribute 및 유니폼

- 정점은 shader로 넘겨줄 수 있는 attribute를 갖고 있음
- 각 정점별로 일정한 유니폼 값도 세이더로 넘겨줄 수 있음
- 세이더는 컴파일되는 외부 프로그램이므로 프로그램 내 모든 변수의 위치를 참조할 수 있어야 함
- 변수의 위치를 가져온 후에는 웹 애플리케이션에서 세이더로 데이터를 전달할 수 있음
- 웹지엘 프로그램 내에서 attribute나 uniform의 위치를 가져오려면 다음 API 사용

Attribute 및 유니폼 2

정점/프래그먼트 세이더 소스에서 찾을 수 있는 attribute 이름

WebGLProgram 객체

```
vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
```

데이터 배열을 attribute로 보낼때 배열 데이터를 활성화함

정점 또는 프래그먼트 세이더 소스에서 찾을 수 있는 attribute 이름

```
gl.enableVertexAttribArray(vertexPositionAttribute);
```

생성된 버퍼 객체

```
gl.bindBuffer(gl.ARRAY_BUFFER, myBuffer);
```

Attribute 성분 개수. Ex. RGB:3, RGBA : 4

Data type

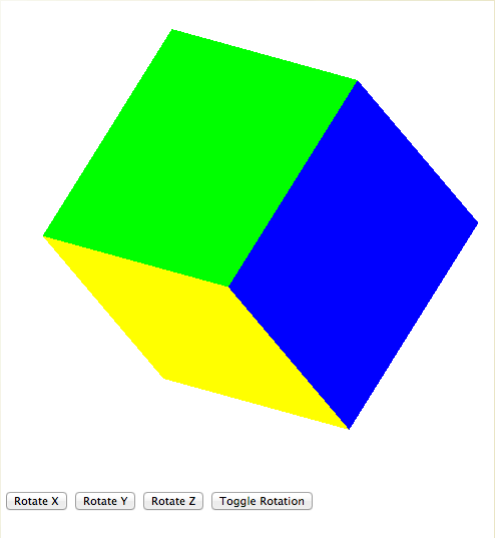
```
gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
```

세이더에게 데이터를 해석하는 법을 알려줌. 세이더는 들어오는 데이터에 대해 전혀 모름. 세이더는 데이터만 접함

void vertexAttribPointer(Glint index, Glint size, Glenum type, Glboolean normalized, Glsize stride, Glintptr offset)

Lab : Drawing Cube

- Render a cube with a different color for each face
- Our example demonstrates:
  - simple object modeling
    - building up 3D objects from geometric primitives
    - building geometric primitives from vertices
  - initializing vertex data
  - organizing data for rendering
  - interactivity
  - animation



Cube data 초기화

- We'll build each cube face from individual triangles
- Need to determine how much storage is required
  - (6 faces)(2 triangles/face)(3 vertices/triangle)

```
var numVertices = 36;
```

- To simplify communicating with GLSL, we'll use a package MV.js which contains a `vec3` object similar to GLSL's `vec3` type

자료 초기화(계속)

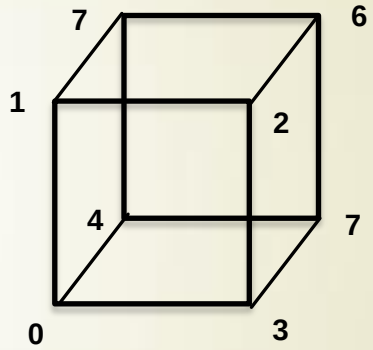
- Before we can initialize our VBO, we need to stage the data
- Our cube has two attributes per vertex
  - position
  - color
- We create two arrays to hold the VBO data

```
var points = [];
var colors = [];
```

Cube data

- Vertices of a unit cube centered at origin
  - sides aligned with axes

```
var vertices = [
        vec4( -0.5, -0.5,  0.5, 1.0 ),
        vec4( -0.5,  0.5,  0.5, 1.0 ),
        vec4(  0.5,  0.5,  0.5, 1.0 ),
        vec4(  0.5, -0.5,  0.5, 1.0 ),
        vec4( -0.5, -0.5, -0.5, 1.0 ),
        vec4( -0.5,  0.5, -0.5, 1.0 ),
        vec4(  0.5,  0.5, -0.5, 1.0 ),
        vec4(  0.5, -0.5, -0.5, 1.0 )
    ];
```



Cube data(계속)

- We'll also set up an array of RGBA colors
- We can use vec3 or vec4 or just JS array

```
var vertexColors = [
        [ 0.0, 0.0, 0.0, 1.0 ],  // black
        [ 1.0, 0.0, 0.0, 1.0 ],  // red
        [ 1.0, 1.0, 0.0, 1.0 ],  // yellow
        [ 0.0, 1.0, 0.0, 1.0 ],  // green
        [ 0.0, 0.0, 1.0, 1.0 ],  // blue
        [ 1.0, 0.0, 1.0, 1.0 ],  // magenta
        [ 0.0, 1.0, 1.0, 1.0 ],  // cyan
        [ 1.0, 1.0, 1.0, 1.0 ]   // white
    ];
```

JS에서의 배열

- A JS array is an object with attributes and methods such as length, push() and pop()
  - fundamentally different from C-style array
  - cannot send directly to WebGL functions
  - use flatten() function to extract data from JS array

```
gl.bufferData( gl.ARRAY_BUFFER, flatten(colors),
  gl.STATIC_DRAW );
```

정점에서 Cube 면 생성하기

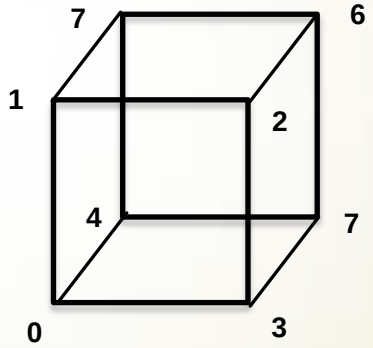
- To simplify generating the geometry, we use a convenience function quad()
  - create two triangles for each face and assigns colors to the vertices

```
function quad(a, b, c, d) {
var indices = [ a, b, c, a, c, d ];
    for ( var i = 0; i < indices.length; ++i ) {
        points.push( vertices[indices[i]] );

        // for vertex colors use
        //colors.push( vertexColors[indices[i]] );

        // for solid colored faces use
        colors.push(vertexColors[a]);
}
```

면에서 cube 생성하기

- Generate 12 triangles for the cube
  - 36 vertices with 36 colors

```
function colorCube() {
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```



정점 attributes 저장하기

- Vertex data must be stored in a Vertex Buffer Object (VBO)
- To set up a VBO we must
  - create an empty by calling gl.createBuffer(); ( )
  - bind a specific VBO for initialization by calling

```
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
```

  - load data into VBO using (for our points)

```
gl.bufferData( gl.ARRAY_BUFFER, flatten(points),
    gl.STATIC_DRAW );
```

Vertex Array Code

Associate shader variables with vertex arrays

```
var cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW );

var vColor = gl.getAttribLocation( program, "vColor" );
gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vColor );

var vBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 3, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
```

Geometric Primitives 그리기

- For contiguous groups of vertices, we can use the simple render function

```
function render()
{
   gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
   gl.drawArrays( gl.TRIANGLES, 0, numVertices );
   requestAnimFrame( render );
}
```

- gl.drawArrays initiates vertex shader
- requestAnimationFrame needed for redrawing if anything is changing
- Note we must clear both the frame buffer and the depth buffer
- Depth buffer used for hidden surface removal
  - enable HSR by gl.enable(gl.GL_DEPTH) in init()

Lab : 실습

1. 주어진 샘플 프로그램을 실행해 보자.
2. Cube 가 y축, z축을 중심으로 회전하게 버튼과 기능을 추가하라.
3. 배경색을 검은 색으로 변경해 보자. 그리고 cube 면색을 변경해보자.
4. 회전 속도를 조절해보자.