

7. Discrete Techniques

Outline

- Buffers
- BitBlt
- Texture Mapping
- WebGL Texture Mapping
- Reflection and Environment Maps
- Bump Maps
- Compositing and Blending
- Imaging Applications
- Framebuffer Objects
- Render to Texture
- Picking by Color
- [Sample Programs](#)

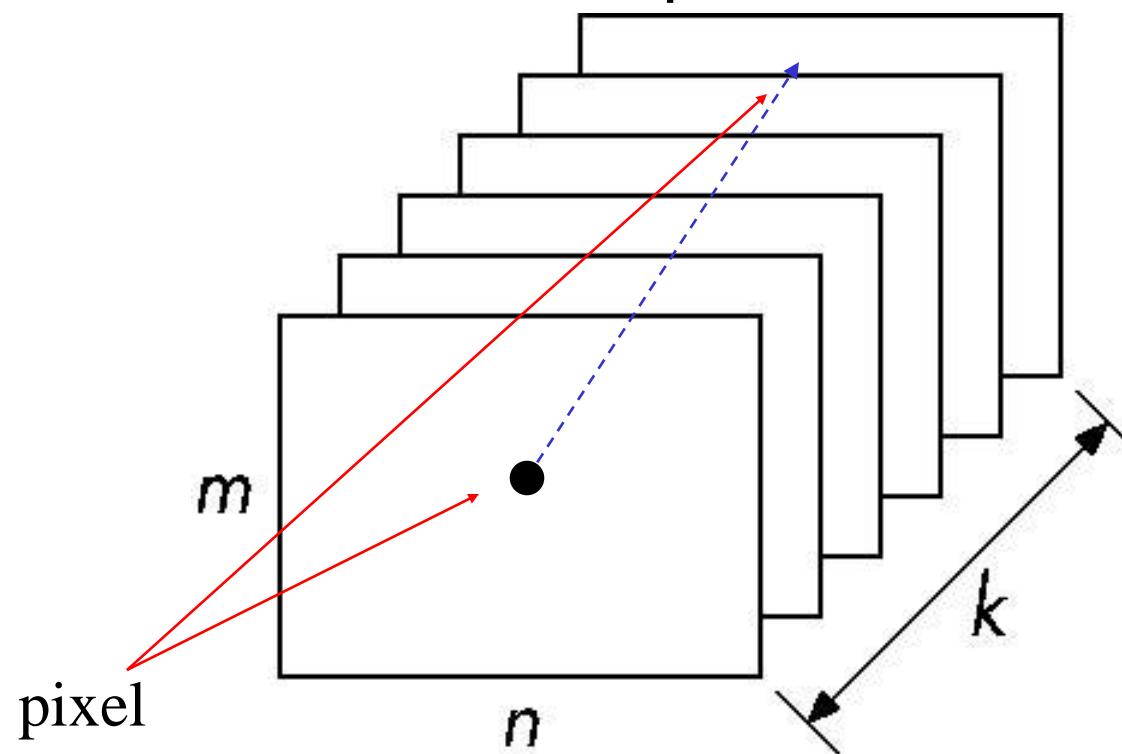
Buffers

Objectives

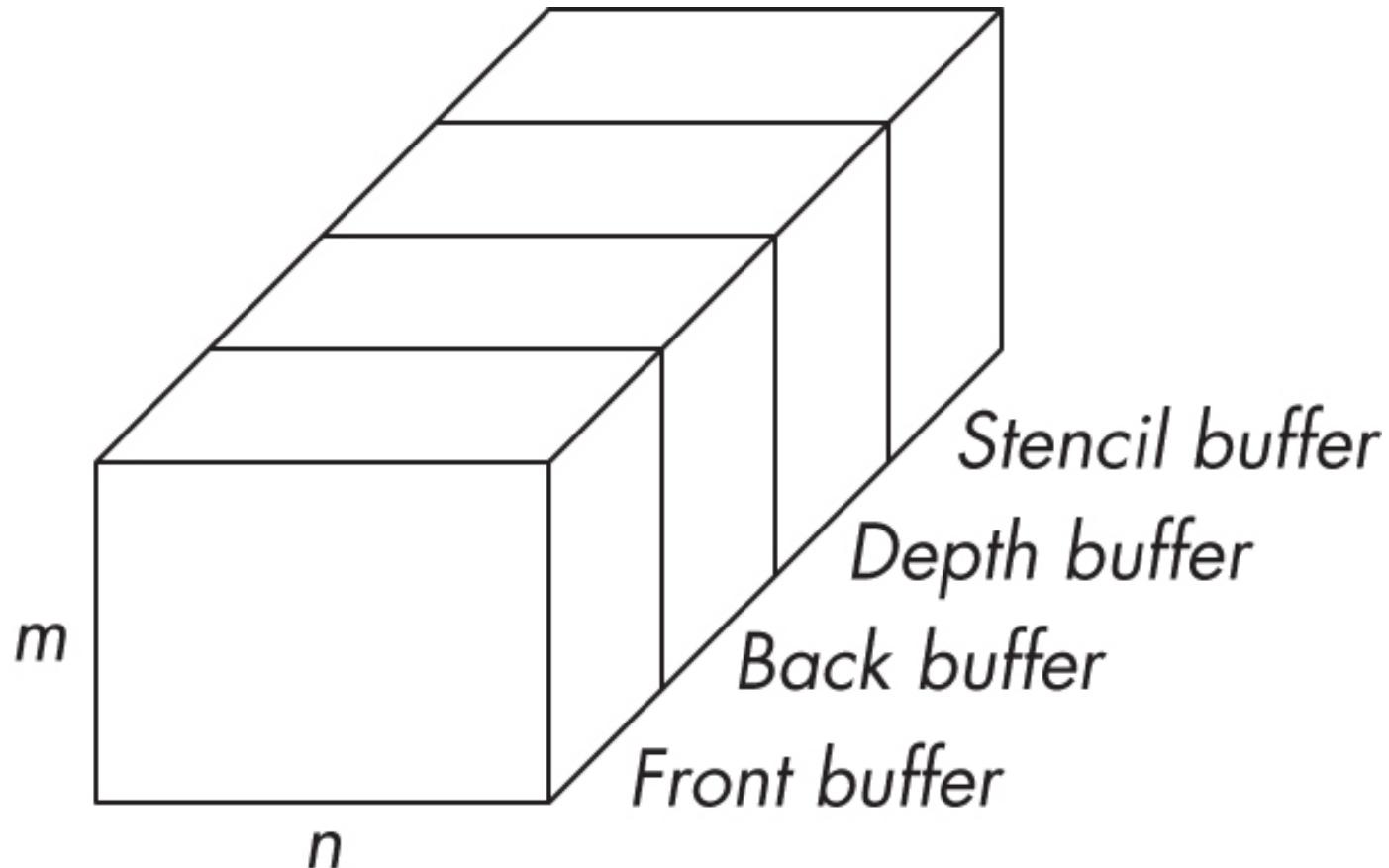
- Introduce additional WebGL buffers
- Reading and writing buffers
- Buffers and Images

Buffer

Define a buffer by its spatial resolution ($n \times m$) and its depth (or precision) k , the number of bits/pixel



WebGL Frame Buffer



Where are the Buffers?

- HTML5 Canvas
 - Default front and back color buffers
 - Under control of local window system
 - Physically on graphics card
- Depth buffer also on graphics card
- Stencil buffer
 - Holds masks
- Most RGBA buffers 8 bits per component
- Latest are floating point (IEEE)

Other Buffers

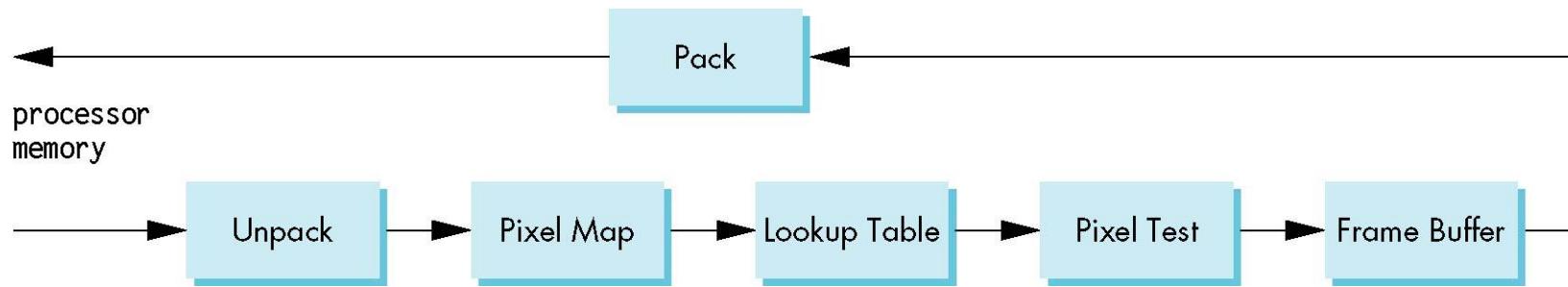
- desktop OpenGL supported other buffers
 - auxiliary color buffers
 - accumulation buffer
 - these were on application side
 - now deprecated
- GPUs have their own or attached memory
 - **texture buffers**
 - **off-screen buffers**
 - not under control of window system
 - may be floating point

Images

- Framebuffer contents are unformatted
 - usually RGB or RGBA
 - one byte per component
 - no compression
- Standard Web Image Formats
 - jpeg, gif, png
- WebGL has no conversion functions
 - Understands standard Web formats for texture images

The (Old) Pixel Pipeline

- OpenGL has a separate pipeline for pixels
 - Writing pixels involves
 - Moving pixels from processor memory to the frame buffer
 - Format conversions
 - Mapping, Lookups, Tests
 - Reading pixels
 - Format conversion



Packing and Unpacking

- Compressed or uncompressed
- Indexed or RGB
- Bit Format
 - little or big endian
- WebGL (and shader-based OpenGL) lacks most functions for packing and unpacking
 - use texture functions instead
 - can implement desired functionality in fragment shaders

Deprecated Functionality

- `glDrawPixels`
- `glCopyPixels`
- `glBitmap`

Buffer Reading

- WebGL can read pixels from the framebuffer with `gl.readPixels`
- Returns only 8 bit RGBA values
- In general, the format of pixels in the frame buffer is different from that of processor memory and these two types of memory reside in different places
 - Need packing and unpacking
 - Reading can be slow
- Drawing through texture functions and off-screen memory (frame buffer objects)

WebGL Pixel Function

```
gl.readPixels(x,y,width,height,format,type,myimage)  
start pixel in frame buffer    size    type of pixels  
                                type of image   pointer to processor  
                                         memory
```

```
var myimage[512*512*4];
```

```
gl.readPixels(0,0, 512, 512, gl.RGBA,  
             gl.UNSIGNED_BYTE, myimage);
```

Render to Texture

- GPUs now include a large amount of **texture memory** that we can write into
- Advantage: **fast** (not under control of window system)
- Using frame buffer objects (FBOs) we can render into texture memory instead of the frame buffer and then read from this memory
 - Image processing
 - GPGPU

BitBlt

Objectives

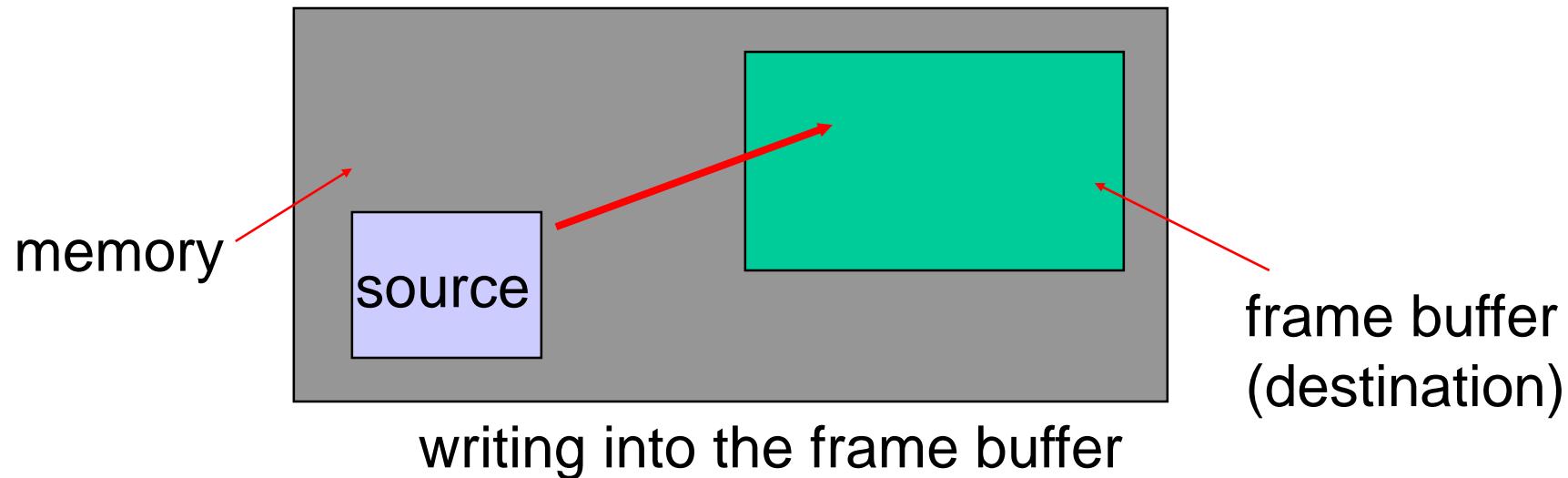
- Introduce reading and writing of blocks of bits or bytes
- Prepare for later discussion compositing and blending

Writing into Buffers

- WebGL does not contain a function for writing bits into frame buffer
 - Use texture functions instead
- We can use the fragment shader to do bit level operations on graphics memory
- Bit Block Transfer (BitBlt) operations act on blocks of bits with a single instruction

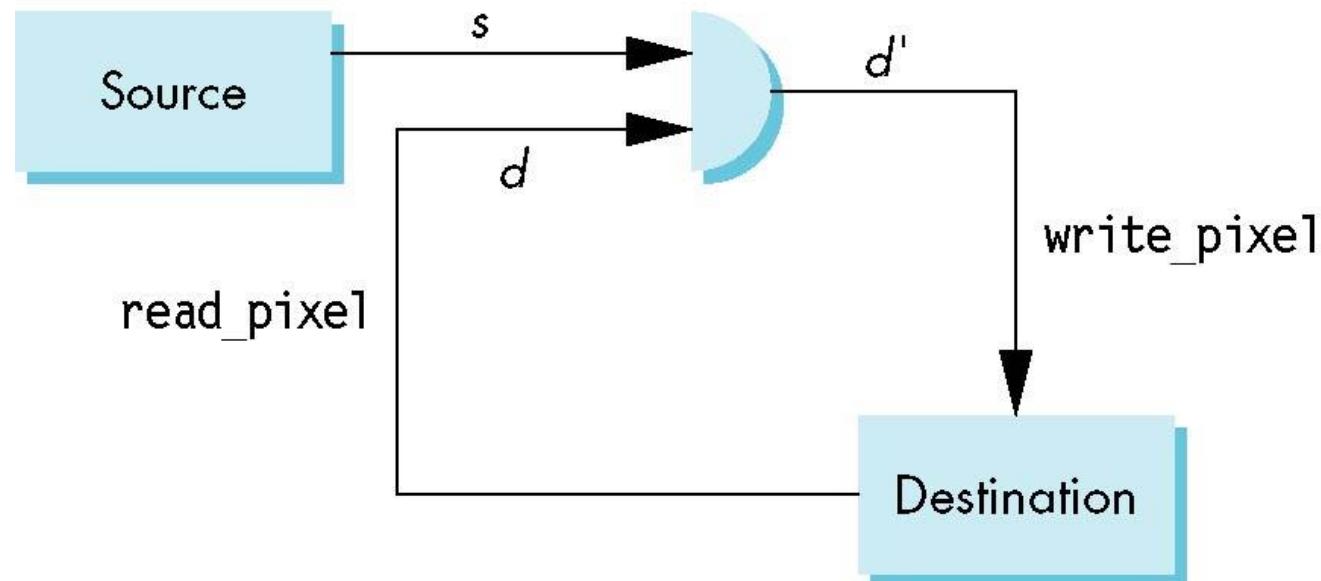
BitBlit

- Conceptually, we can consider all of memory as a large **two-dimensional array of pixels**
- We read and write rectangular block of pixels
- The frame buffer is part of this memory



Writing Model

Read destination pixel before writing source



Bit Writing Modes

- Source and destination bits are combined bitwise
- 16 possible functions (one per column in table)

s	d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

XOR mode

- XOR is especially useful for swapping blocks of memory such as menus that are stored off screen

If S represents screen and M represents a menu
the sequence

$$S \leftarrow S \oplus M$$

$$M \leftarrow S \oplus M$$

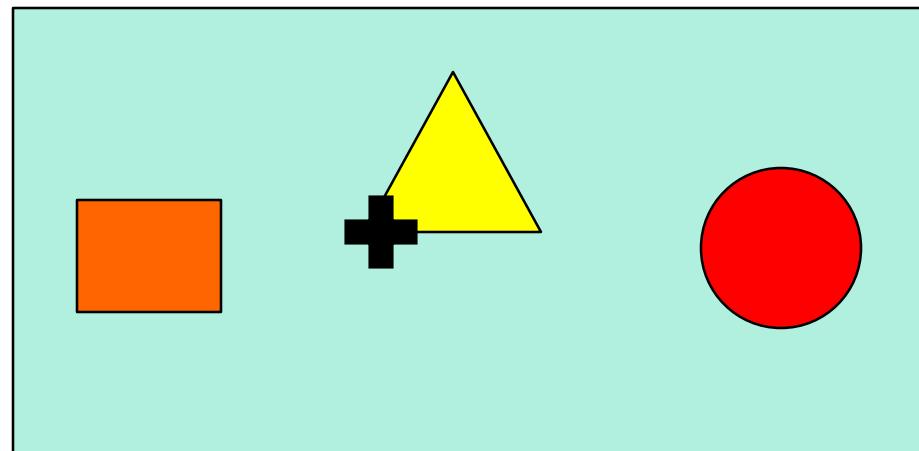
$$S \leftarrow S \oplus M$$

swaps S and M

- Same strategy used for rubber band lines and cursors

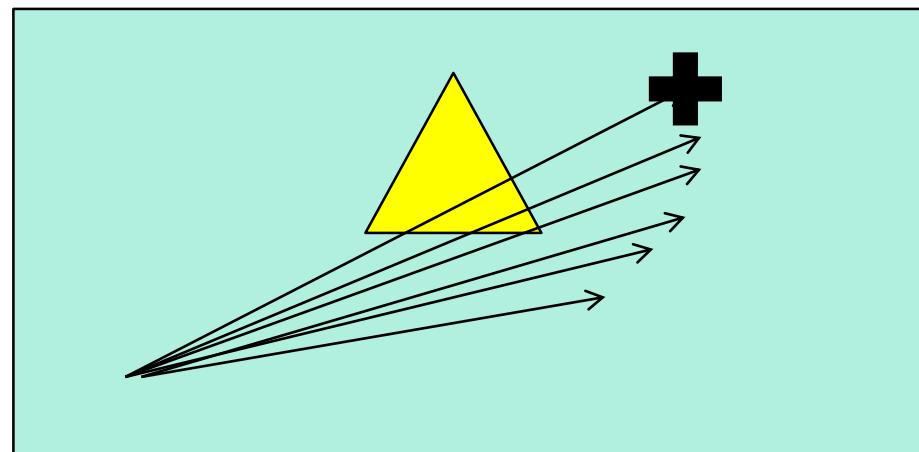
Cursor Movement

- Consider what happens as we move a cursor across the display
- We cover parts of objects
- Must return to original colors when cursor moves away



Rubber Band Line

- Fix one point
- Draw line to location of cursor
- Must return state of crossed objects when line moves



Texture Mapping

Objectives

- Introduce Mapping Methods
 - Texture Mapping
 - Environment Mapping
 - Bump Mapping
- Consider basic strategies
 - Forward vs backward mapping
 - Point sampling vs area averaging

The Limits of Geometric Modeling

- Although graphics cards can render over 10 million polygons per second, that number is insufficient for many phenomena
 - Clouds
 - Grass
 - Terrain
 - Skin

Modeling an Orange

- Consider the problem of modeling an orange (the fruit)
- Start with an orange-colored sphere
 - Too simple
- Replace sphere with a more complex shape
 - Does not capture surface characteristics (small dimples)
 - Takes too many polygons to model all the dimples

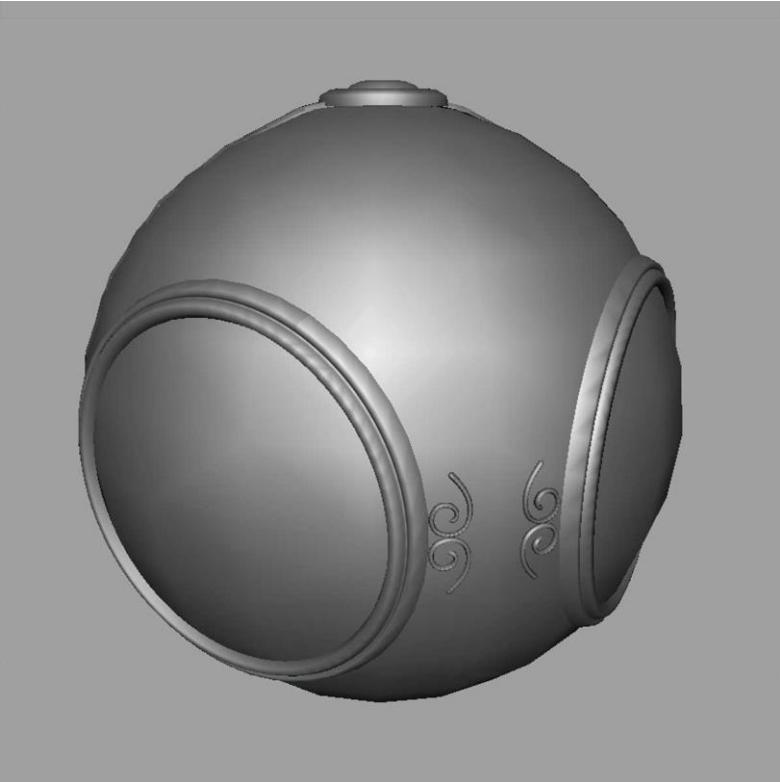
Modeling an Orange (2)

- Take a picture of a real orange, scan it, and “paste” onto simple geometric model
 - This process is known as **texture mapping**
- Still might not be sufficient because resulting surface will be smooth
 - Need to **change local shape**
 - **Bump mapping**

Three Types of Mapping

- Texture Mapping
 - Uses images to fill inside of polygons
- Environment (reflection mapping)
 - Uses a picture of the environment for texture maps
 - Allows simulation of highly specular surfaces
- Bump mapping
 - Emulates altering normal vectors during the rendering process

Texture Mapping



geometric model

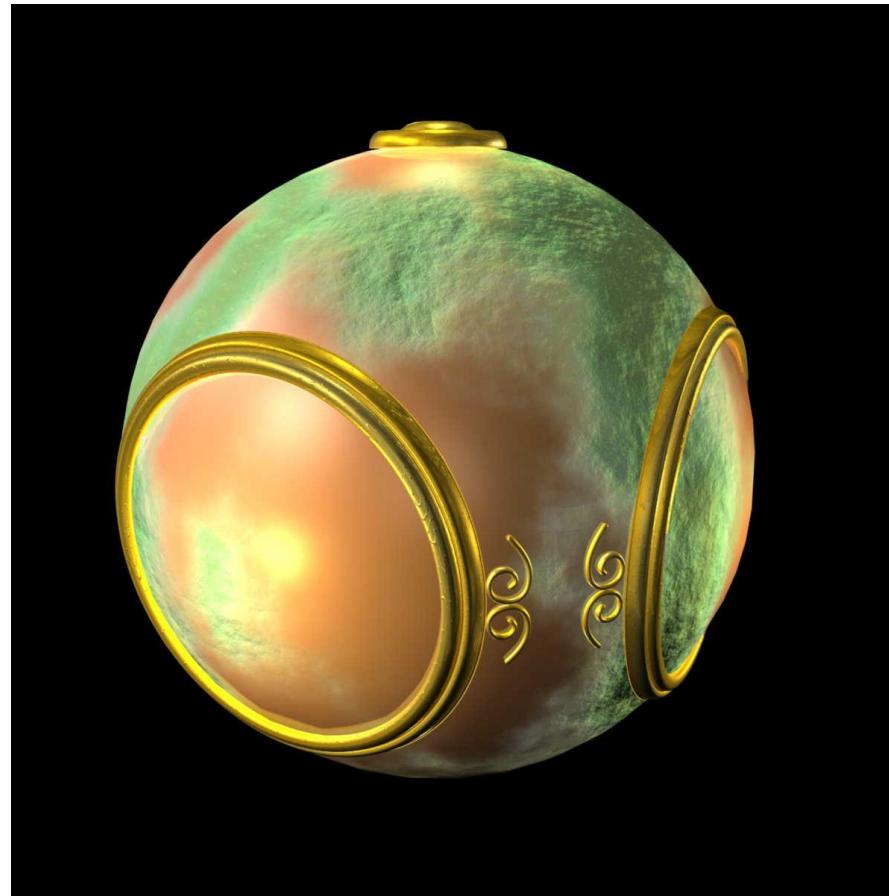


texture mapped

Environment Mapping

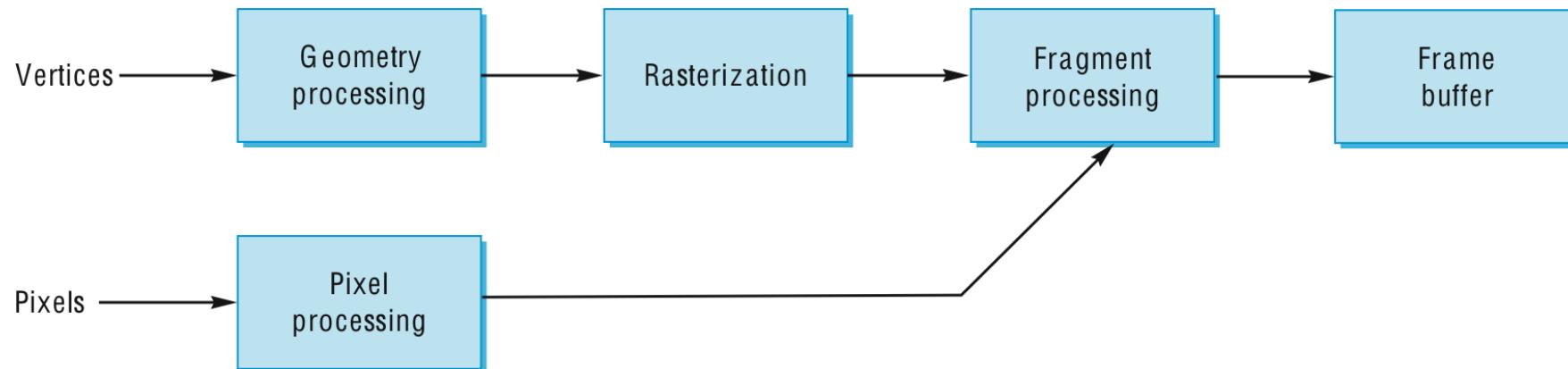


Bump Mapping



Where does mapping take place?

- Mapping techniques are implemented **at the end of the rendering pipeline**
 - Very efficient because few polygons make it past the clipper



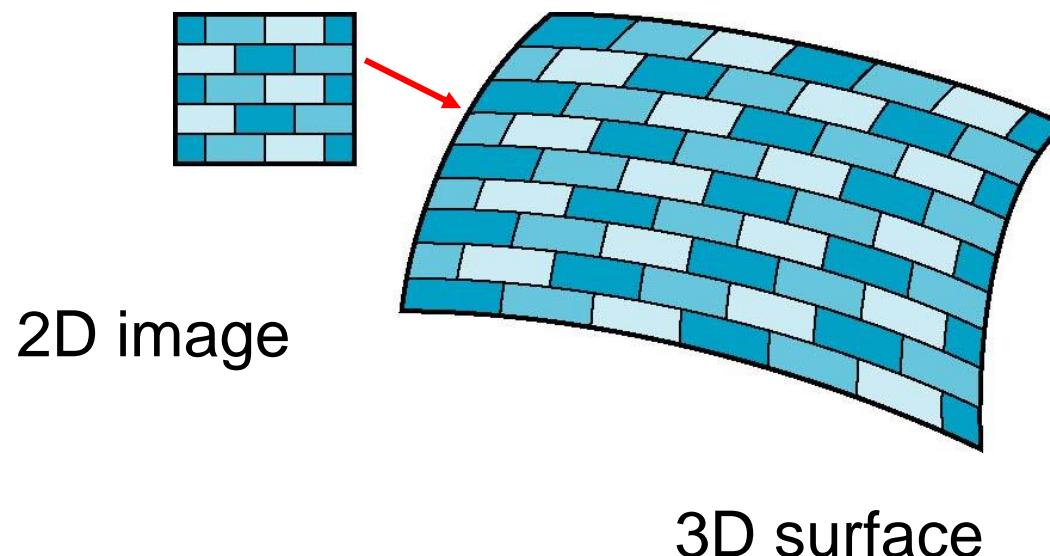
Texture Mapping

Objectives

- Basic mapping strategies
 - Forward vs backward mapping
 - Point sampling vs area averaging

Is it simple?

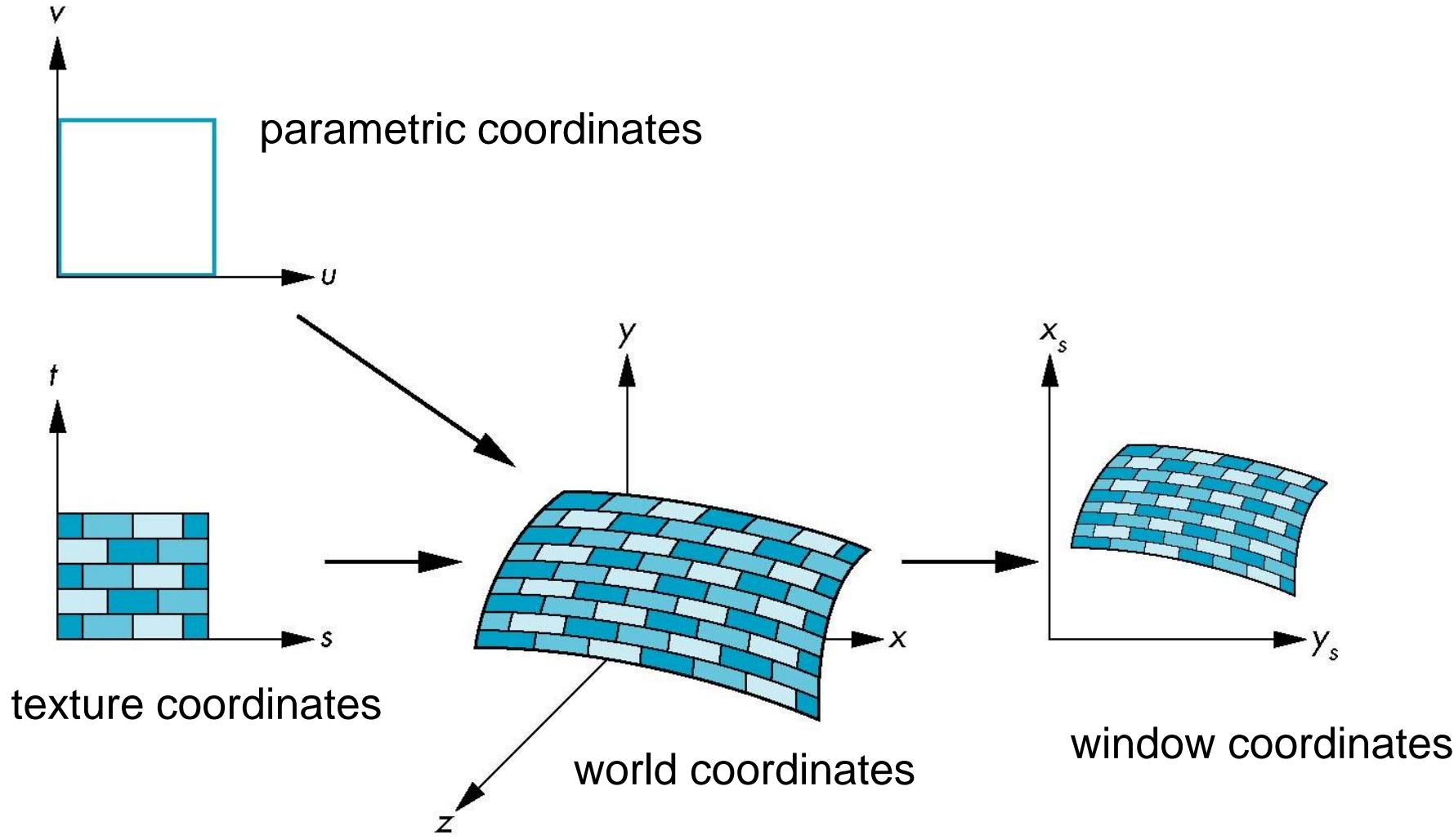
- Although the idea is simple---map an image to a surface--there are 3 or 4 coordinate systems involved



Coordinate Systems

- Parametric coordinates
 - May be used to model curves and surfaces
- Texture coordinates
 - Used to identify points in the image to be mapped
- Object or World Coordinates
 - Conceptually, where the mapping takes place
- Window Coordinates
 - Where the final image is really produced

Texture Mapping



Mapping Functions

- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface

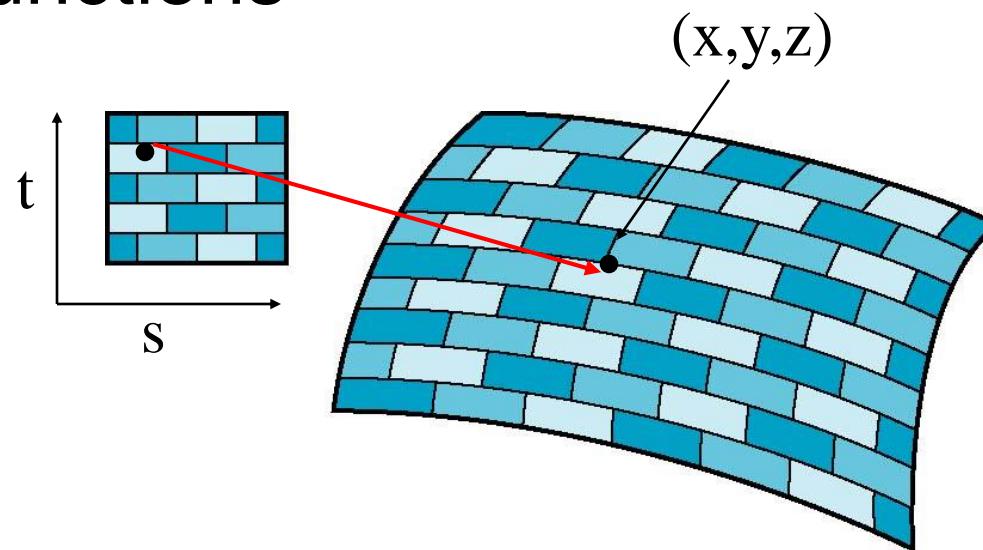
- Appear to need three functions

$$x = x(s,t)$$

$$y = y(s,t)$$

$$z = z(s,t)$$

- But we really want to go the other way

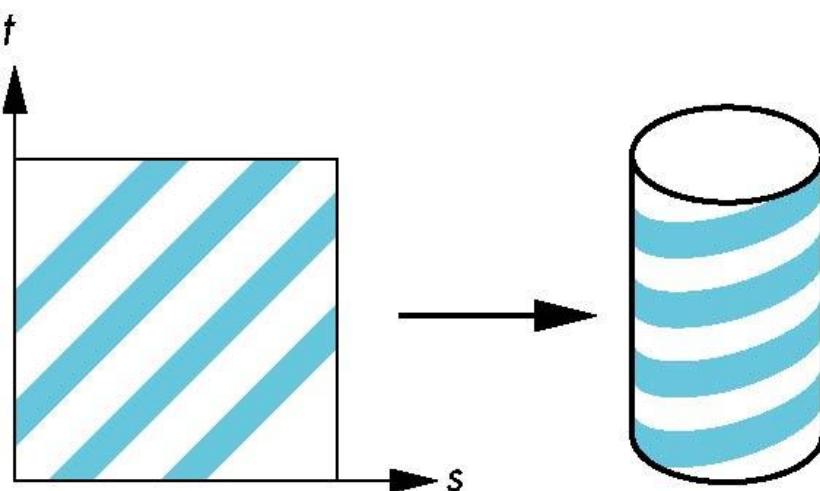


Backward Mapping

- We really want to go backwards
 - Given a pixel, we want to know to which point on an object it corresponds
 - Given a point on an object, we want to know to which point in the texture it corresponds
- Need a map of the form
$$s = s(x, y, z)$$
$$t = t(x, y, z)$$
- Such functions are difficult to find in general

Two-part mapping

- One solution to the mapping problem is to **first map the texture to a simple intermediate surface**
- Example: map to cylinder



Cylindrical Mapping

parametric cylinder

$$x = r \cos 2\pi u$$

$$y = r \sin 2\pi u$$

$$z = v/h$$

maps rectangle in u, v space to cylinder
of **radius r and height h** in world coordinates

$$s = u$$

$$t = v$$

maps from texture space

Spherical Map

We can use a parametric sphere

$$x = r \cos 2\pi u$$

$$y = r \sin 2\pi u \cos 2\pi v$$

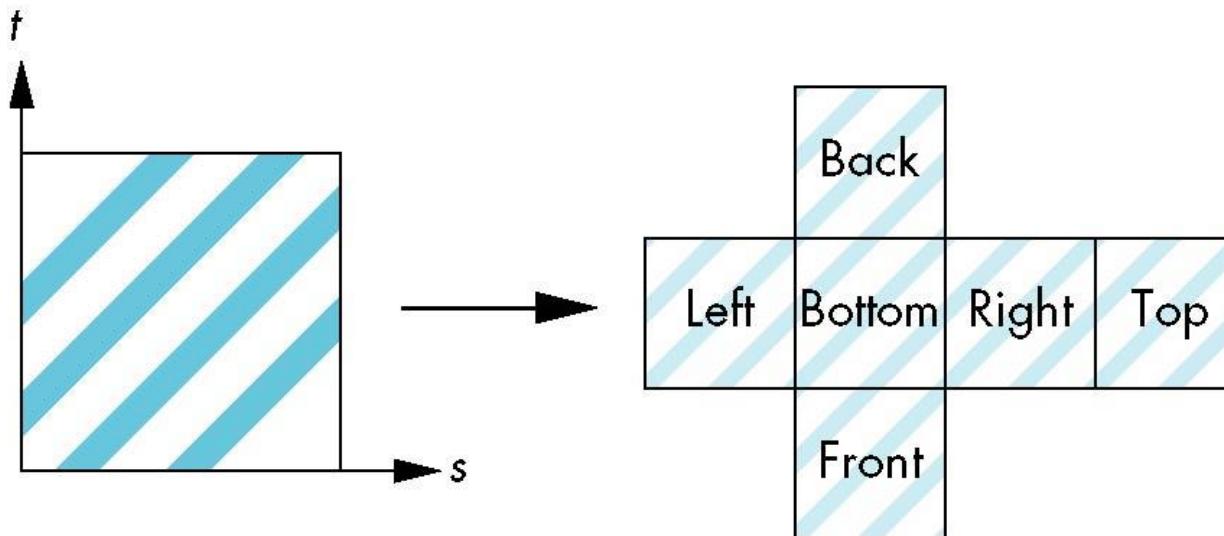
$$z = r \sin 2\pi u \sin 2\pi v$$

in a similar manner to the cylinder
but have to decide where to put
the distortion

Spheres are used in **environmental maps**

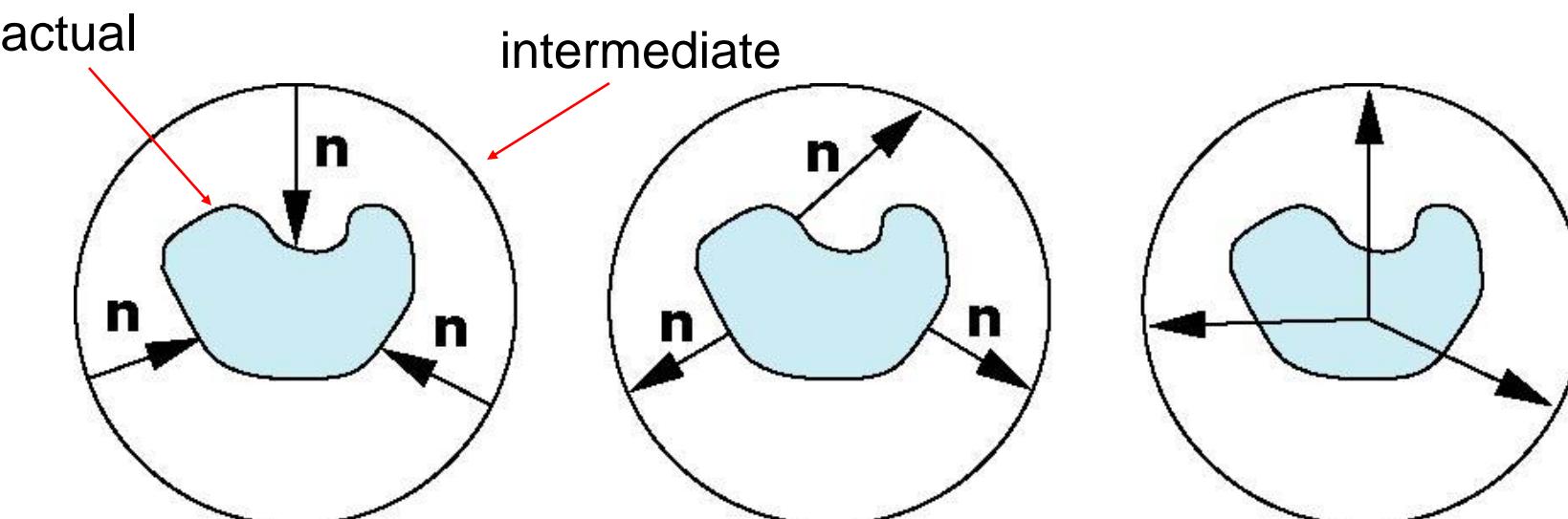
Box Mapping

- Easy to use with simple orthographic projection
- Also used in environment maps



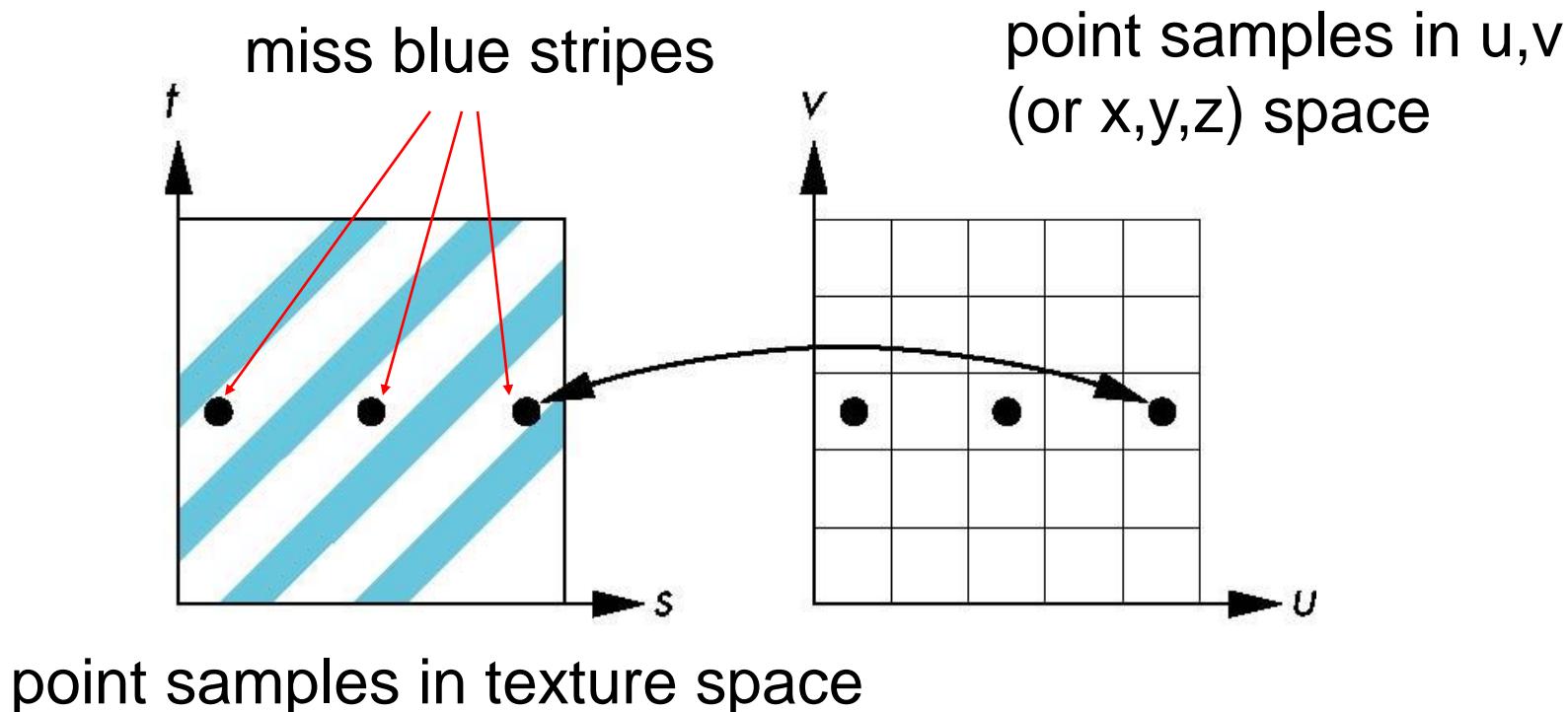
Second Mapping

- Map from intermediate object to actual object
 - Normals from intermediate to actual
 - Normals from actual to intermediate
 - Vectors from center of intermediate



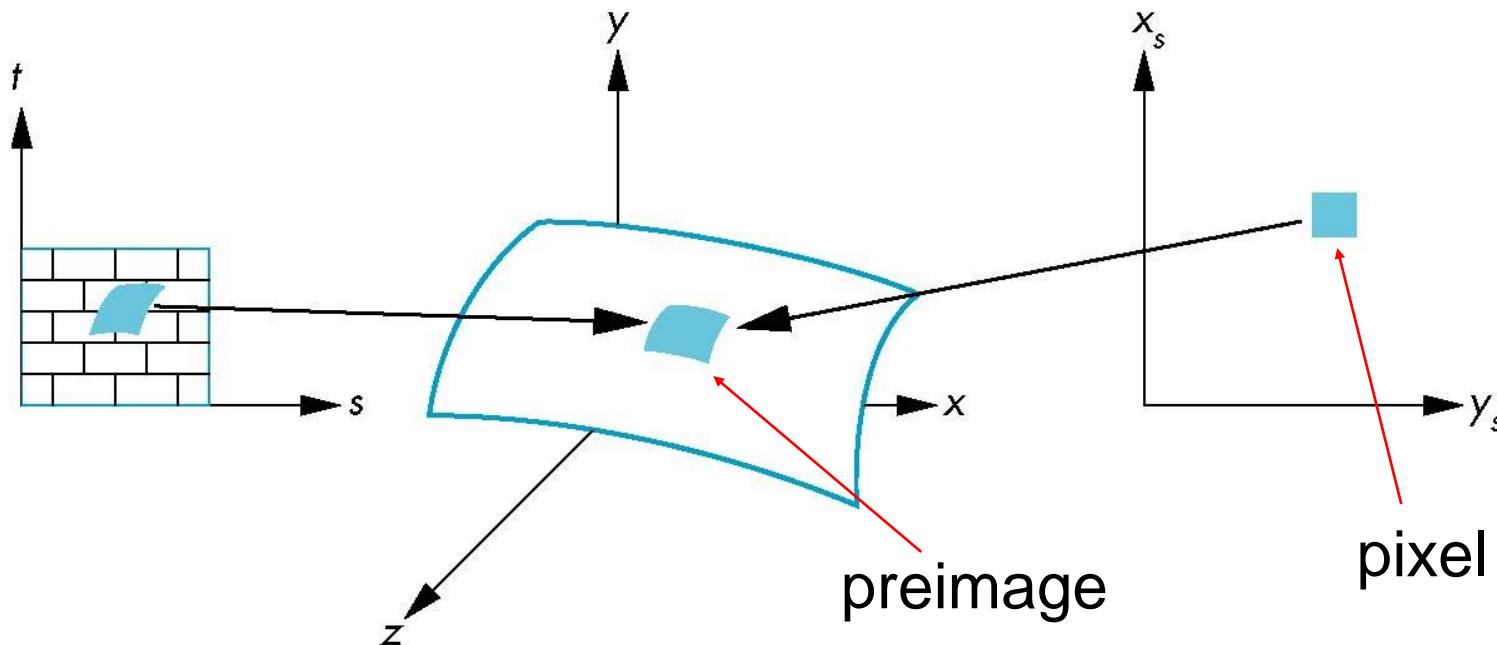
Aliasing

- Point sampling of the texture can lead to aliasing errors



Area Averaging

A better but slower option is to use *area averaging*



Note that *preimage* of pixel is curved

WebGL Texture Mapping I

Objectives

- Introduce WebGL texture mapping
 - two-dimensional texture maps
 - assigning texture coordinates
 - forming texture images

Basic Strategy

Three steps to applying a texture

1. specify the texture

- read or generate image
- assign to texture
- enable texturing

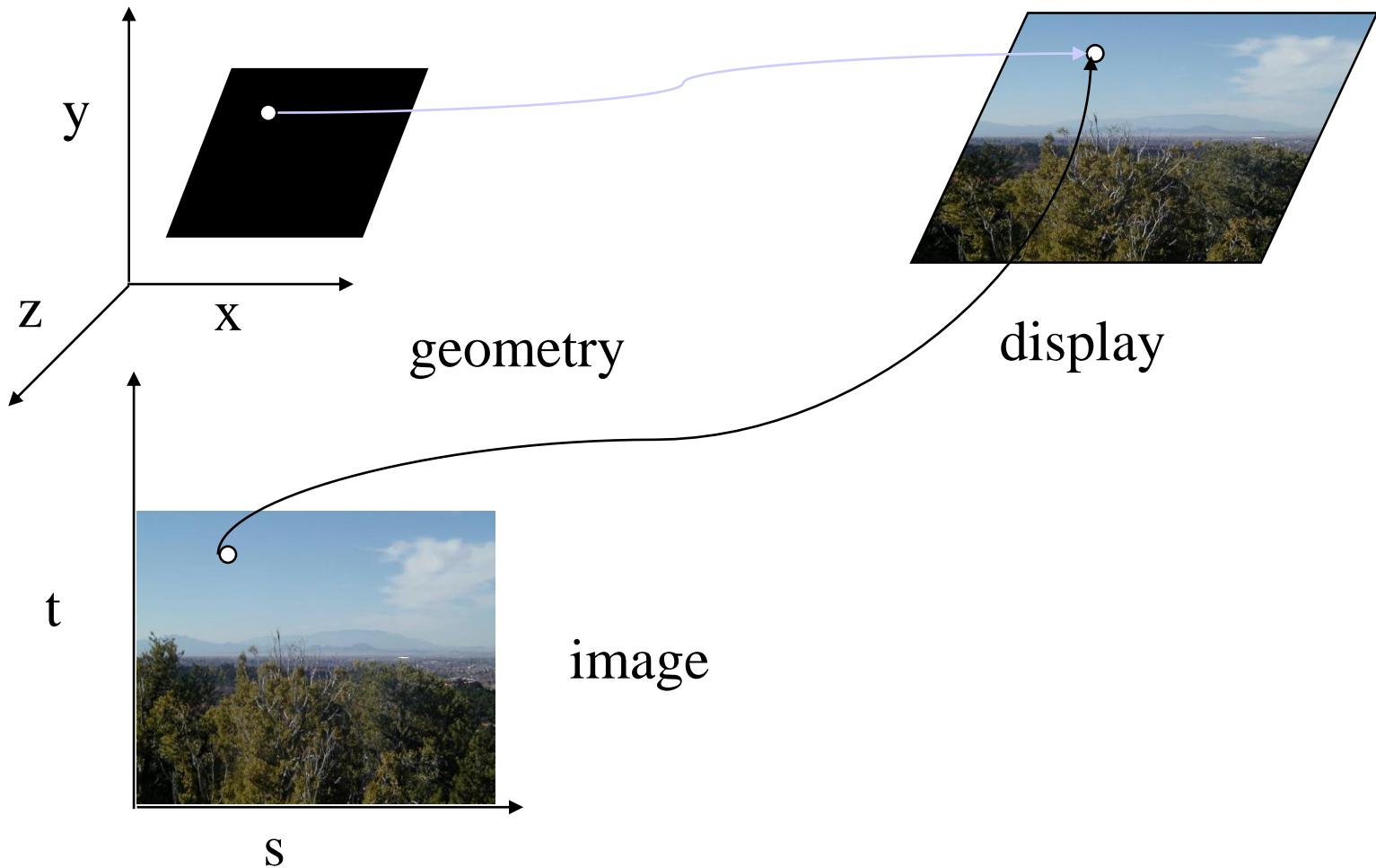
2. assign texture coordinates to vertices

- Proper mapping function is left to application

3. specify texture parameters

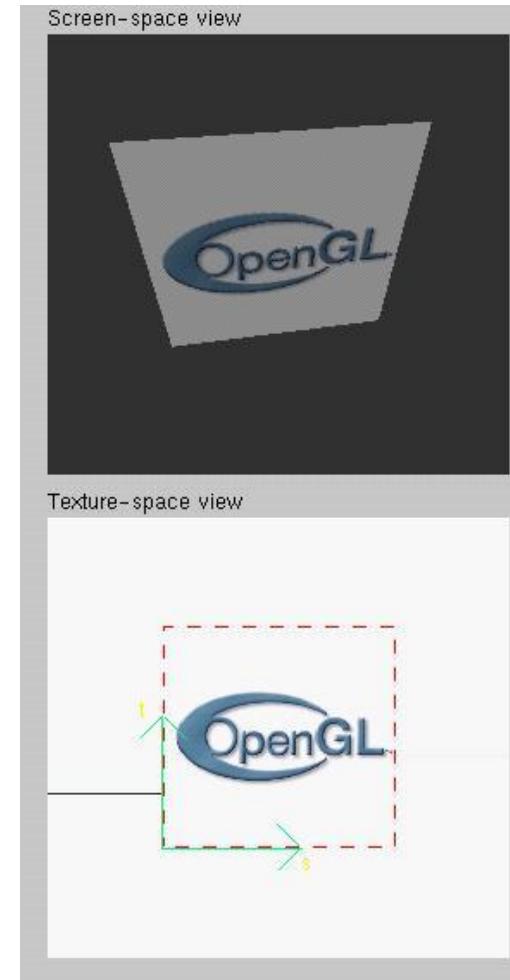
- wrapping, filtering

Texture Mapping



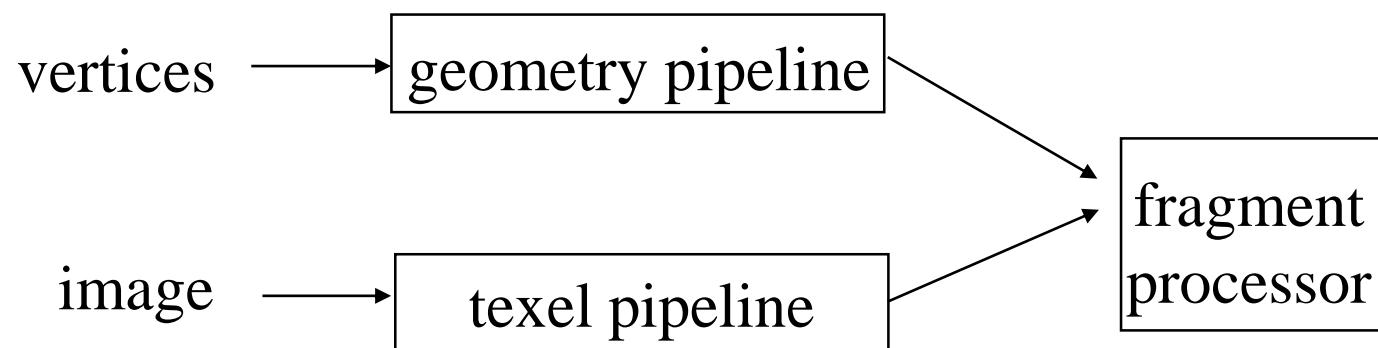
Texture Example

- The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective



Texture Mapping and the WebGL Pipeline

- Images and geometry flow through separate pipelines that join during fragment processing
 - “complex” textures do not affect geometric complexity



Specifying a Texture Image

- Define a **texture image** from an array of *texels* (texture elements) in CPU memory
- Use an image in a standard format such as JPEG
 - Scanned image
 - Generate by application code
- **WebGL supports only 2 dimensional texture maps**
 - no need to enable as in desktop OpenGL
 - desktop OpenGL supports 1-4 dimensional texture maps

Define Image as a Texture

```
glTexImage2D( target, level, components,  
    w, h, border, format, type, texels );
```

target: type of texture, e.g. `GL_TEXTURE_2D`

level: used for mipmapping (discussed later)

components: elements per texel

w, **h**: width and height of `texels` in pixels

border: used for smoothing (discussed later)

format and **type**: describe texels

texels: pointer to texel array

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0,  
    GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```

A Checkerboard Image

```
var image1 = new Uint8Array(4*texSize*texSize);
for ( var i = 0; i < texSize; i++ ) {
    for ( var j = 0; j <texSize; j++ ) {
        var patchx = Math.floor(i/(texSize/numChecks));
        var patchy = Math.floor(j/(texSize/numChecks));
        if(patchx%2 ^ patchy%2) c = 255;
        else c = 0;
        //c = 255*((i & 0x8) == 0) ^ ((j & 0x8) == 0))
        image1[4*i*texSize+4*j] = c;
        image1[4*i*texSize+4*j+1] = c;
        image1[4*i*texSize+4*j+2] = c;
        image1[4*i*texSize+4*j+3] = 255;
    }
}
```

Using a GIF image

```
// specify image in JS file
```

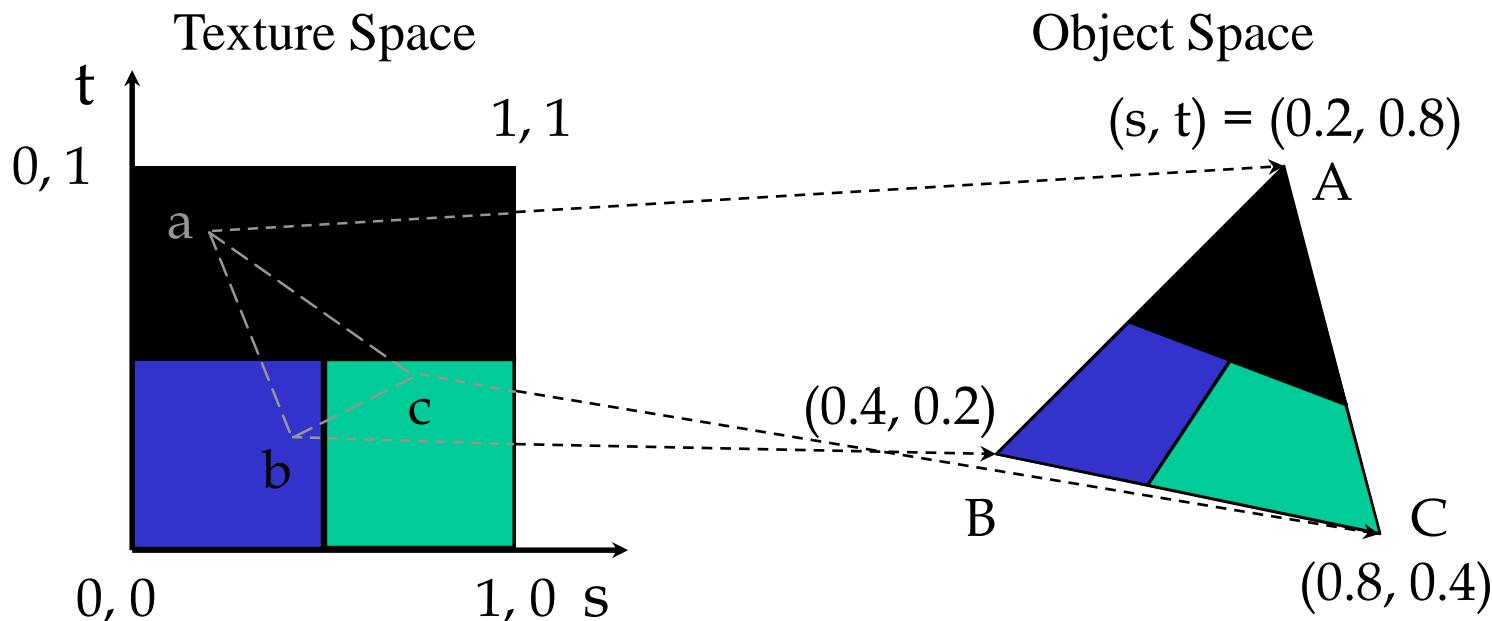
```
var image = new Image();
image.onload = function() {
    configureTexture( image );
}
image.src = "SA2011_black.gif"
```

```
// or specify image in HTML file with <img> tag
// <img id = "texImage" src = "SA2011_black.gif"></img>
```

```
var image = document.getElementById("texImage")
window.onload = configureTexture( image );
```

Mapping a Texture

- Based on **parametric texture coordinates**
- Specify as a 2D vertex attribute



Cube Example

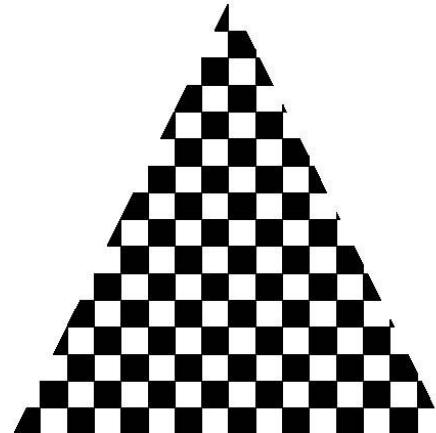
```
var texCoord = [  
    vec2(0, 0),  
    vec2(0, 1),  
    vec2(1, 1),  
    vec2(1, 0)  
];  
  
function quad(a, b, c, d) {  
    pointsArray.push(vertices[a]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[0]);  
  
    pointsArray.push(vertices[b]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[1]);  
    // etc
```

Interpolation

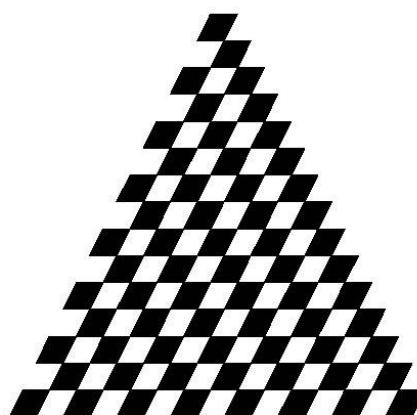
WebGL uses interpolation to find proper texels from specified texture coordinates

Can be **distortions**

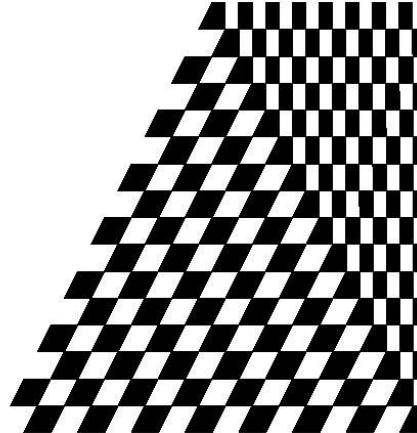
good selection
of tex coordinates



poor selection
of tex coordinates



texture stretched
over trapezoid
showing effects of
bilinear interpolation



WebGL Texture Mapping I

Objectives

- Introduce the WebGL texture functions and options
 - texture objects
 - texture parameters
 - example code

Using Texture Objects

1. specify textures in texture objects
2. set texture filter
3. set texture function
4. set texture wrap mode
5. set optional perspective correction hint
6. bind texture object
7. enable texturing
8. supply texture coordinates for vertex
 - coordinates can also be generated

Texture Parameters

- WebGL has a variety of parameters that determine how texture is applied
 - Wrapping parameters determine what happens if s and t are outside the $(0,1)$ range
 - Filter modes allow us to use area averaging instead of point samples
 - Mipmapping allows us to use textures at multiple resolutions
 - Environment parameters determine how texture mapping interacts with shading

Wrapping Mode

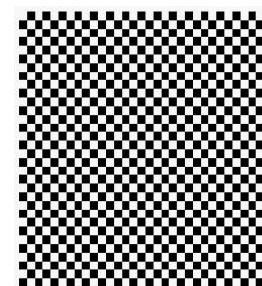
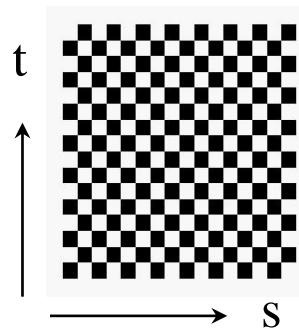
Clamping: if $s,t > 1$ use 1, if $s,t < 0$ use 0

gl.CLAMP

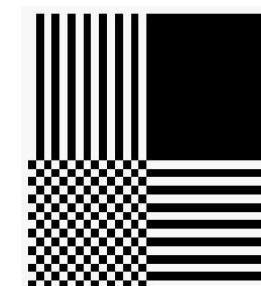
Wrapping: use s,t modulo 1

gl.REPEAT

```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP )
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.REPEAT)
```



gl.REPEAT
wrapping

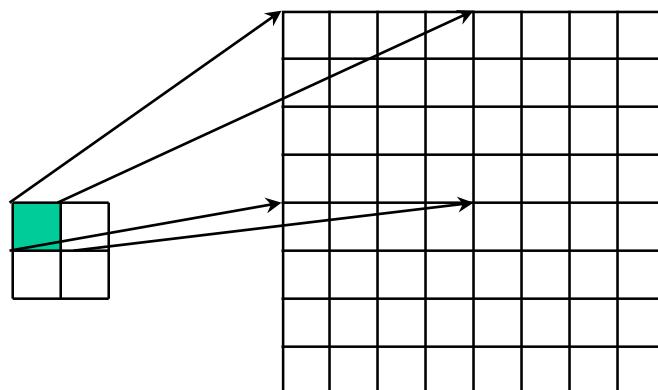


gl.CLAMP
wrapping

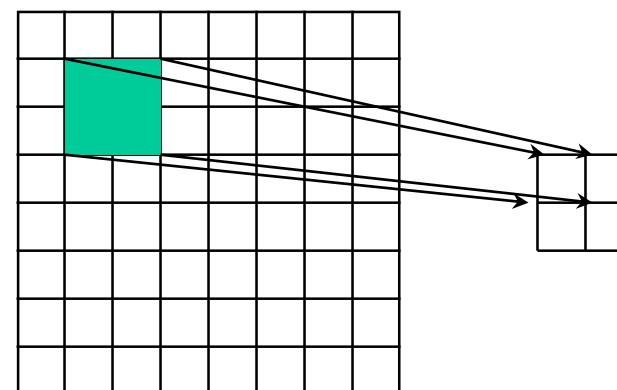
Magnification and Minification

More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

Can use point sampling (nearest texel) or linear filtering (2 x 2 filter) to obtain texture values



Texture
Magnification



Texture
Minification

Filter Modes

Modes determined by

`gl.texParameteri(target, type, mode)`

```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
```

```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
```

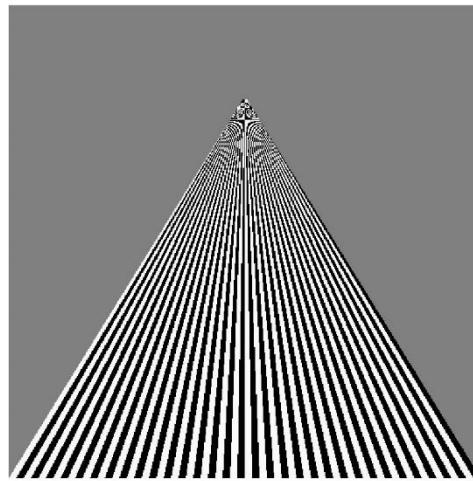
Mipmapped Textures

- *Mipmapping* allows for **prefiltered texture maps** of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition

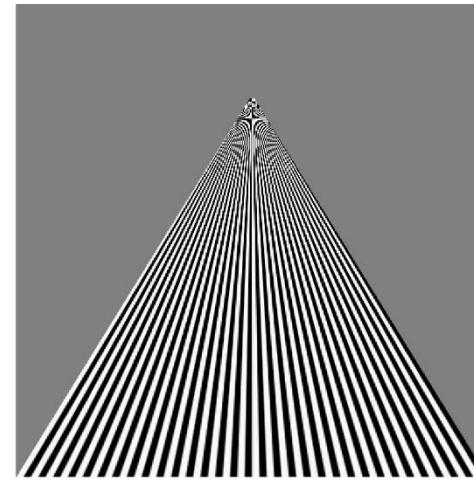
```
gl.texImage2D(gl.TEXTURE_2D, level, ... )
```

Example

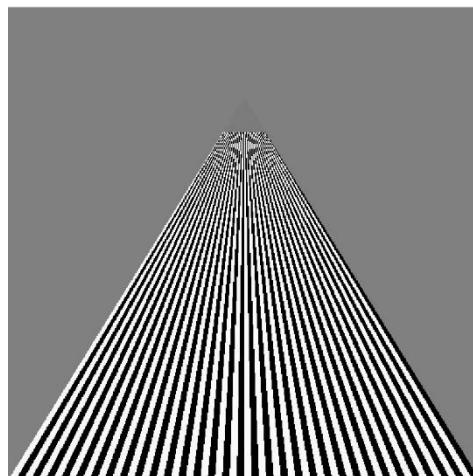
point
sampling



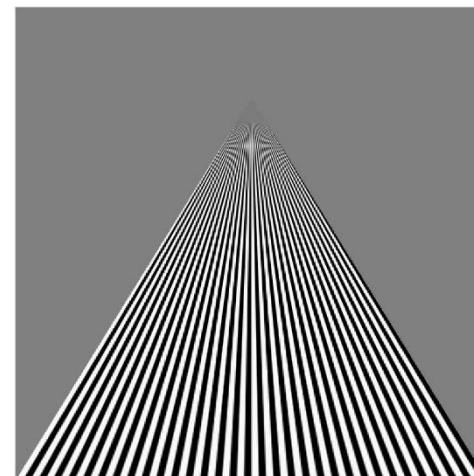
linear
filtering



mipmapped
point
sampling



mipmapped
linear
filtering



Applying Textures

- Texture can be applied in many ways
 - texture fully determines color
 - modulated with a computed color
 - blended with an environmental color
- Fixed function pipeline has a function `glTexEnv` to set mode
 - deprecated
 - can get all desired functionality via fragment shader
- Can also use multiple texture units

Other Texture Features

- Environment Maps
 - Start with image of environment through a wide angle lens
 - Can be either a real scanned image or an image created in OpenGL
 - Use this texture to generate a spherical map
 - Alternative is to use a cube map
- Multitexturing
 - Apply a sequence of textures through cascaded texture units

Applying Textures

- Textures are applied during fragments shading by a **sampler**
- Samplers **return a texture color** from a texture object

```
varying vec4 color;           //color from rasterizer
varying vec2 texCoord;        //texture coordinate from rasterizer
uniform sampler2D texture; //texture object from application
```

```
void main() {
    gl_FragColor = color * texture2D( texture, texCoord );
}
```

Vertex Shader

- Usually vertex shader will output texture coordinates to be rasterized
- Must do all other standard tasks too
 - Compute vertex position
 - Compute vertex color if needed

```
attribute vec4 vPosition; //vertex position in object coordinates
```

```
attribute vec4 vColor; //vertex color from application
```

```
attribute vec2 vTexCoord; //texture coordinate from application
```

```
varying vec4 color; //output color to be interpolated
```

```
varying vec2 texCoord; //output tex coordinate to be interpolated
```

A Checkerboard Image

```
var image1 = new Uint8Array(4*texSize*texSize);
for ( var i = 0; i < texSize; i++ ) {
    for ( var j = 0; j <texSize; j++ ) {
        var patchx = Math.floor(i/(texSize/numChecks));
        var patchy = Math.floor(j/(texSize/numChecks));
        if(patchx%2 ^ patchy%2) c = 255;
            else c = 0;
        //c = 255*((i & 0x8) == 0) ^ ((j & 0x8) == 0))
        image1[4*i*texSize+4*j] = c;
        image1[4*i*texSize+4*j+1] = c;
        image1[4*i*texSize+4*j+2] = c;
        image1[4*i*texSize+4*j+3] = 255;
    }
}
```

Cube Example

```
var texCoord = [  
    vec2(0, 0),  
    vec2(0, 1),  
    vec2(1, 1),  
    vec2(1, 0)  
];  
  
function quad(a, b, c, d) {  
    pointsArray.push(vertices[a]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[0]);  
  
    pointsArray.push(vertices[b]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[1]);  
    // etc
```

Texture Object

```
function configureTexture( image ) {  
    var texture = gl.createTexture();  
    gl.bindTexture( gl.TEXTURE_2D, texture );  
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);  
    gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image );  
    gl.generateMipmap( gl.TEXTURE_2D );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR);  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );  
    gl.activeTexture( gl.TEXTURE0);  
    gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);  
}
```

Linking with Shaders

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

// Set the value of the fragment shader texture sampler variable
// ("texture") to the the appropriate texture unit. In this case,
// zero for GL_TEXTURE0 which was previously set by calling
// gl.activeTexture().

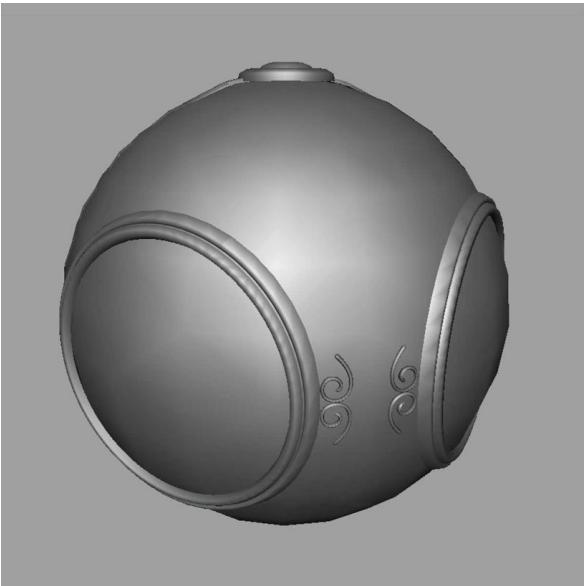
gl.uniform1i( glGetUniformLocation(program, "texture"), 0 );
```

Reflection and Environment Maps

Objectives

- Texture Mapping Applications
- Reflection (Environment) Maps
 - Cube Maps
 - Spherical Maps
- Bump Maps

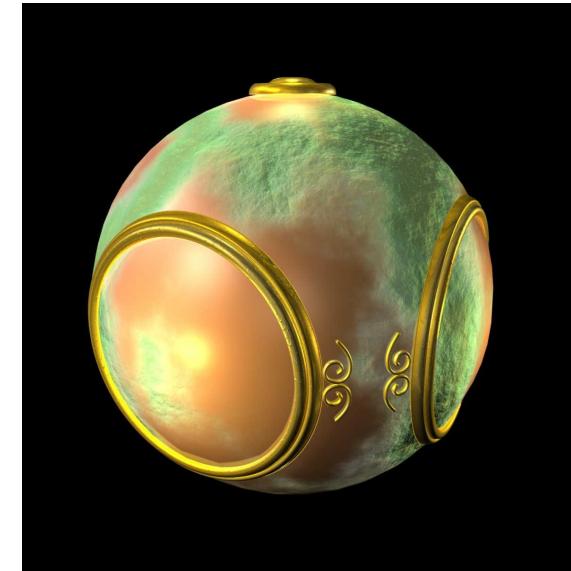
Mapping Variations



smooth shading



environment
mapping

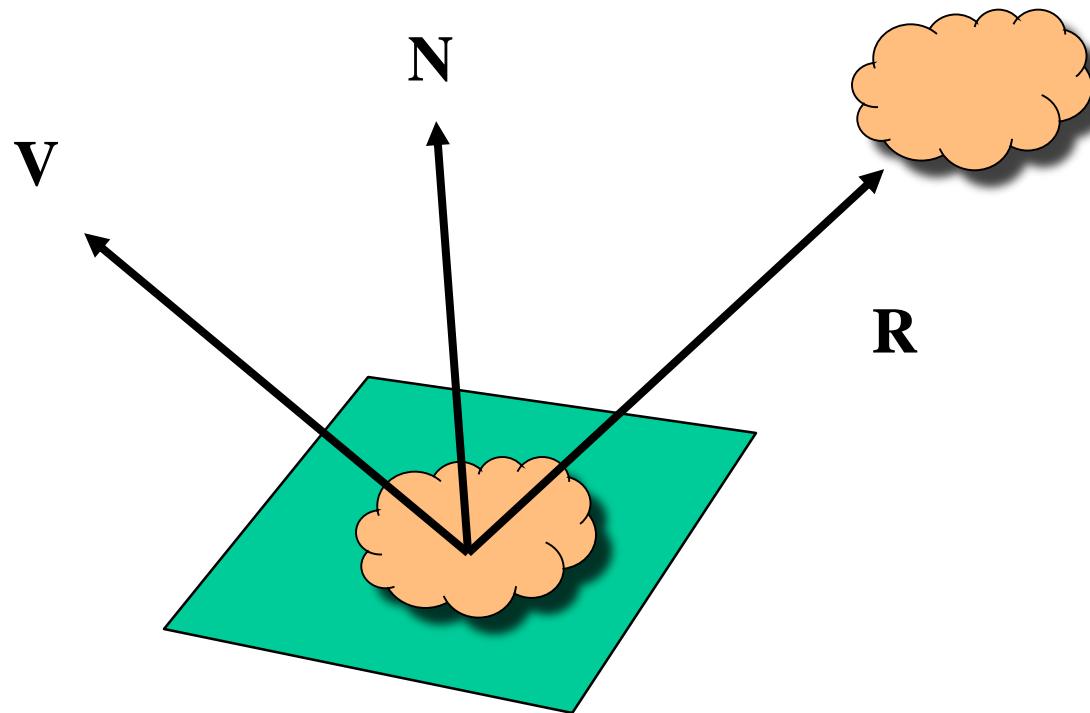


bump mapping

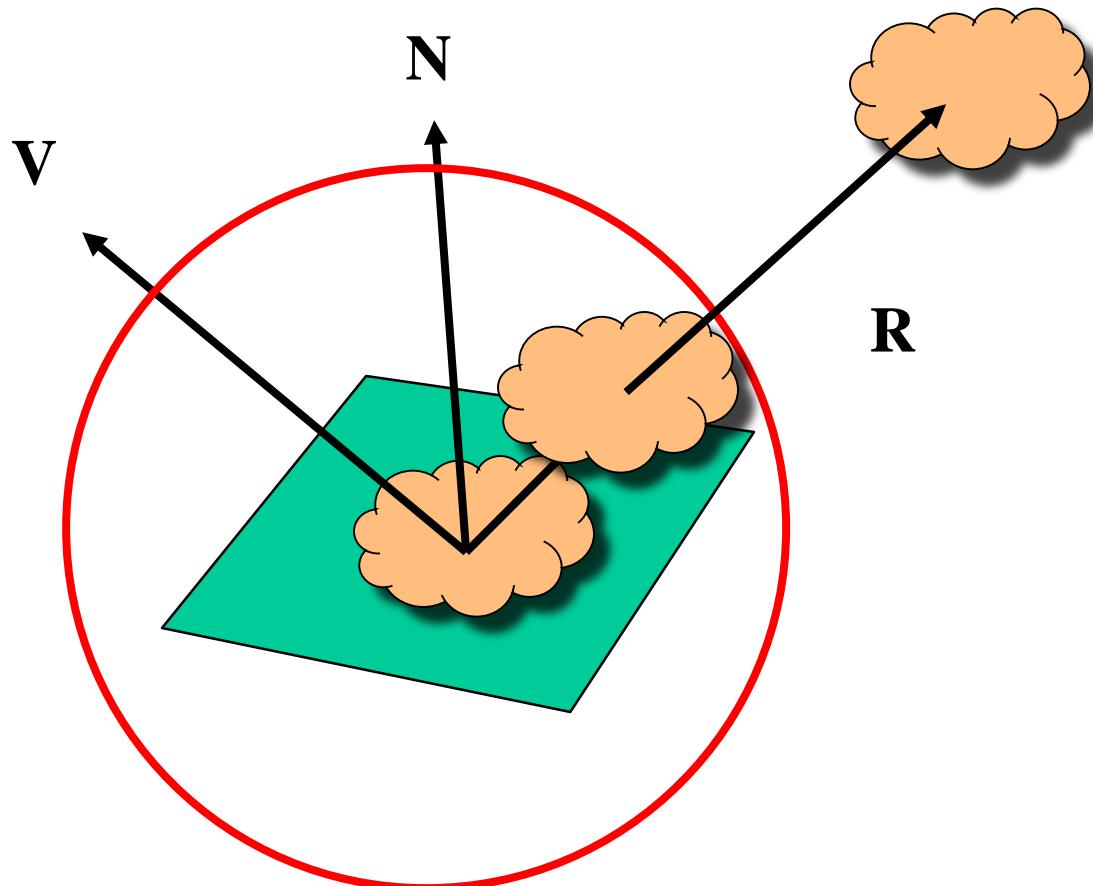
Environment Mapping

- Environmental (reflection) mapping is way to create the appearance of highly reflective surfaces without ray tracing which requires global calculations
- Introduced in movies such as The Abyss and Terminator 2
- Prevalent in video games
- It is a form of texture mapping

Reflecting the Environment



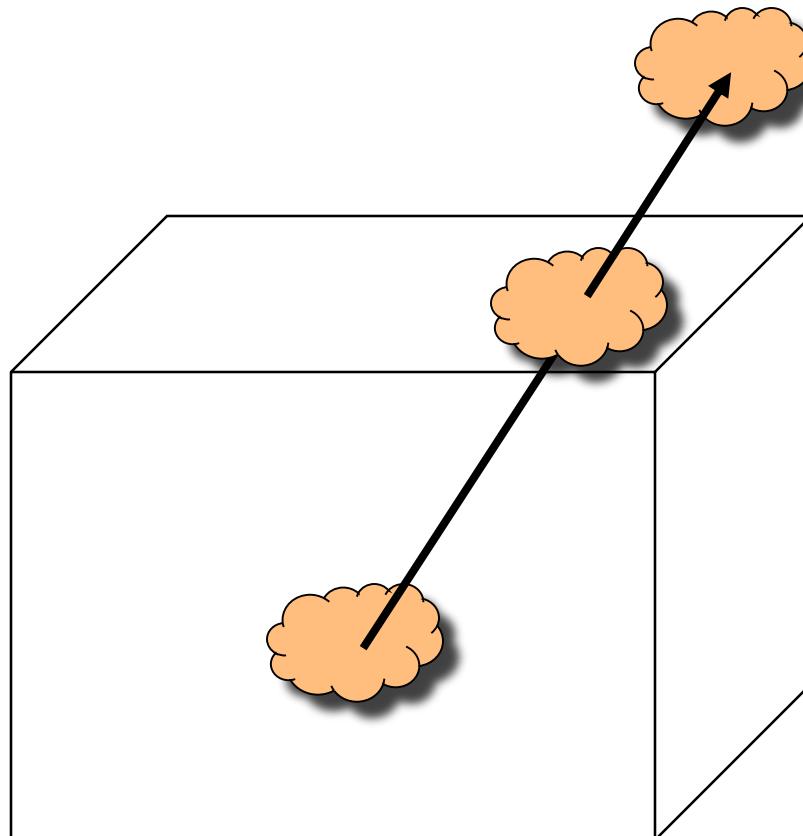
Mapping to a Sphere



Hemisphere Map as a Texture

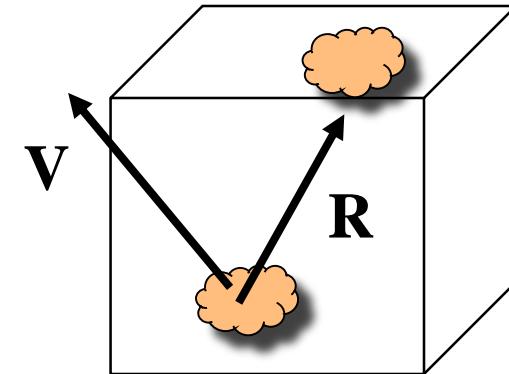
- If we map all objects to hemisphere, we cannot tell if they are on the sphere or anywhere else along the reflector
- Use the map on the sphere as a texture that can be mapped onto the object
- Can use other surfaces as the intermediate
 - Cube maps
 - Cylinder maps

Cube Map



Indexing into Cube Map

- Compute $\mathbf{R} = 2(\mathbf{N} \cdot \mathbf{V})\mathbf{N} - \mathbf{V}$
- Object at origin
- Use largest magnitude component of \mathbf{R} to determine face of cube
- Other two components give texture coordinates



OpenGL Implementation

- WebGL supports only cube maps
 - desktop OpenGL also supports sphere maps
- First must form map
 - Use images from a real camera
 - Form images with WebGL
- Texture map it to object

Cube Maps

- We can form a cube map texture by defining six 2D texture maps that correspond to the sides of a box
- Supported by WebGL through cubemap sampler

```
vec4 texColor = textureCube(mycube, texcoord);
```
- **Texture coordinates must be 3D**
 - usually are given by the vertex location so we don't need compute separate tex coords

Environment Maps with Shaders

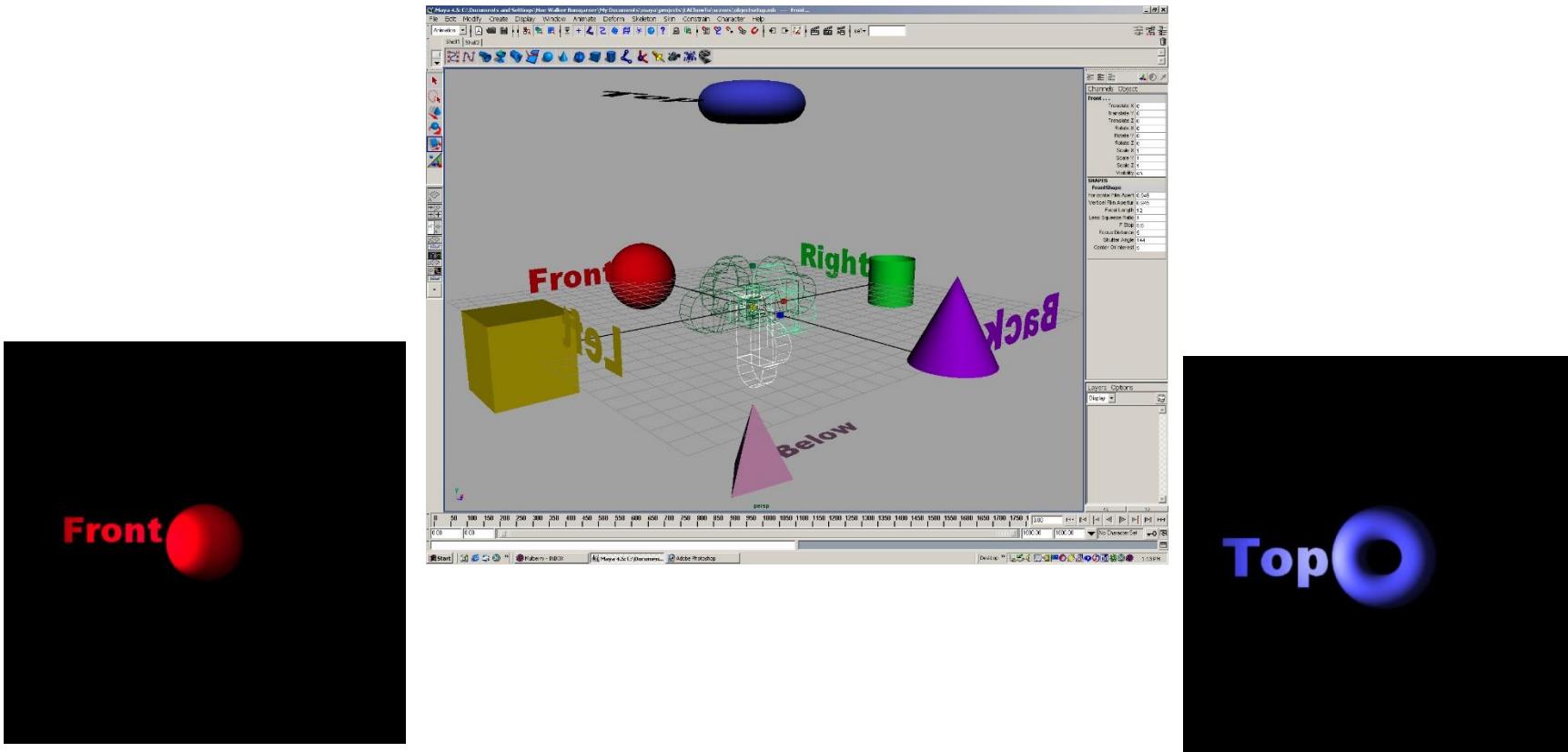
- Environment maps are usually computed in world coordinates which can differ from object coordinates because of the modeling matrix
 - May have to keep track of modeling matrix and pass it to the shaders as a uniform variable
- Can also use reflection map or refraction map for effects such as simulating water

Issues

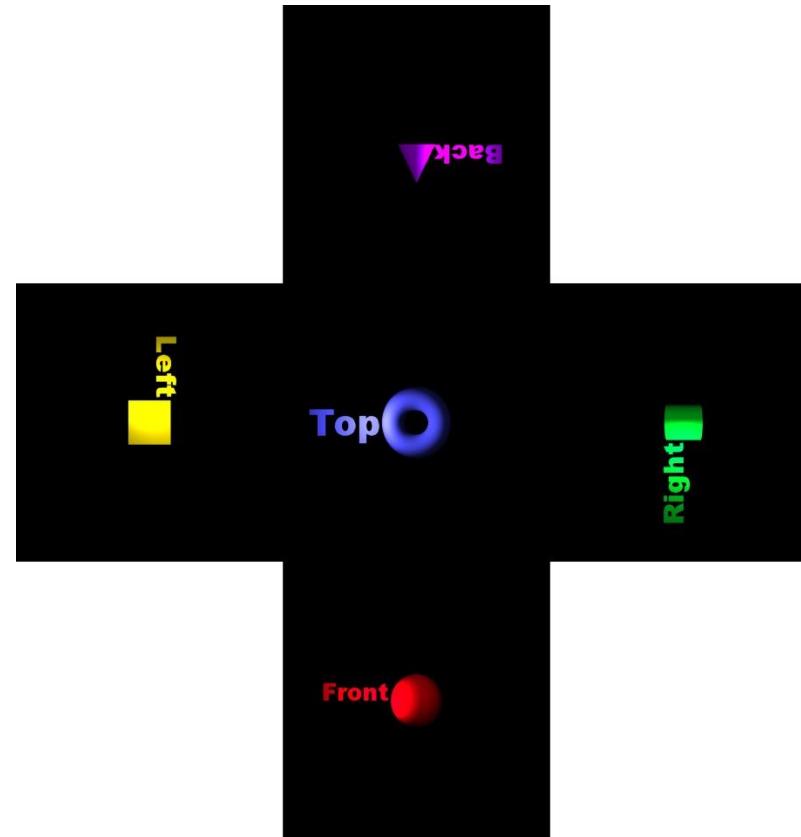
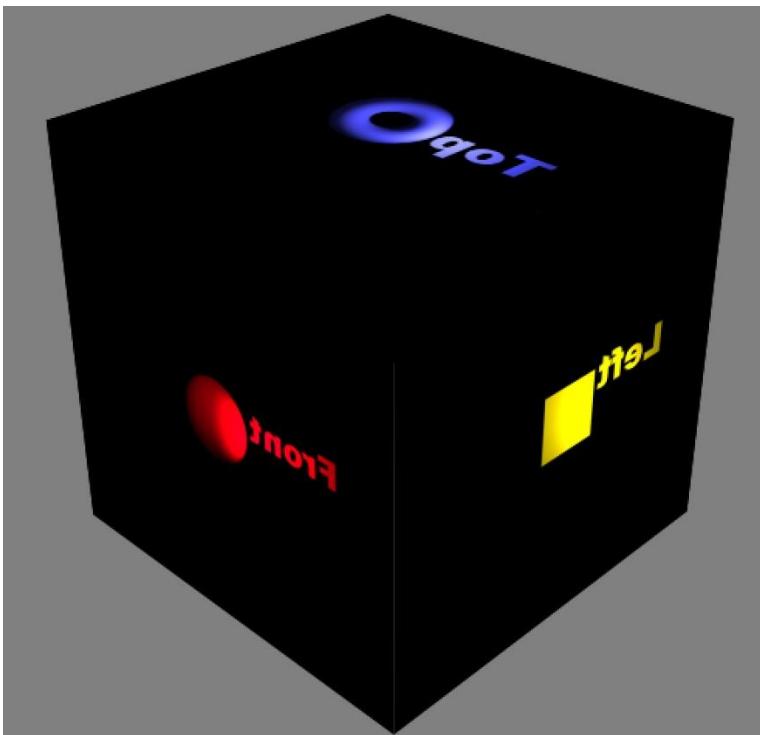
- Must assume environment is very far from object (equivalent to the difference between near and distant lights)
- Object cannot be concave (no self reflections possible)
- No reflections between objects
- Need a reflection map for each object
- Need a new map if viewer moves

Forming Cube Map

- Use six cameras, each with a 90 degree angle of view



vs Cube Image



Doing it in WebGL

```
gl.textureMap2D(  
    gl.TEXTURE_CUBE_MAP_POSITIVE_X,  
    level, rows, columns, border, gl.RGBA,  
    gl.UNSIGNED_BYTE, image1)
```

- Same for other five images
- Make one texture object out of the six images

Example

- Consider rotating cube inside a cube that reflects the color of the walls
- Each wall is a solid color (red, green, blue, cyan, magenta, yellow)
 - Each face of room can be a texture of one texel

```
var red      = new Uint8Array([255,  0,  0, 255]);
var green    = new Uint8Array([ 0, 255,  0, 255]);
var blue     = new Uint8Array([ 0,  0, 255, 255]);
var cyan     = new Uint8Array([ 0, 255, 255, 255]);
var magenta  = new Uint8Array([255,  0, 255, 255]);
var yellow   = new Uint8Array([255, 255,  0, 255]);
```

Texture Object

```
cubeMap = gl.createTexture();
gl.bindTexture(gl.TEXTURE_CUBE_MAP, cubeMap);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_X, 0, gl.RGBA,
             1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, red);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_X, 0, gl.RGBA,
             1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, green);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_Y, 0, gl.RGBA,
             1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, blue);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_Y, 0, gl.RGBA,
             1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, cyan);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_Z, 0, gl.RGBA,
             1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, yellow);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_Z, 0, gl.RGBA,
             1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, magenta);
gl.activeTexture( gl.TEXTURE0 );
gl.uniform1i(gl.getUniformLocation(program, "texMap"), 0);
```

Vertex Shader

```
varying vec3 R;  
attribute vec4 vPosition;  
attribute vec4 vNormal;  
uniform mat4 modelViewMatrix;  
uniform mat4 projectionMatrix;  
uniform vec3 theta;  
void main(){  
    vec3 angles = radians( theta );  
    // compute rotation matrices rx, ry, rz here  
    mat4 ModelViewMatrix = modelViewMatrix * rz * ry * rx;  
    gl_Position = projectionMatrix * ModelViewMatrix * vPosition;  
    vec4 eyePos = ModelViewMatrix * vPosition;  
    vec4 N = ModelViewMatrix * vNormal;  
    R = reflect(eyePos.xyz, N.xyz); }
```

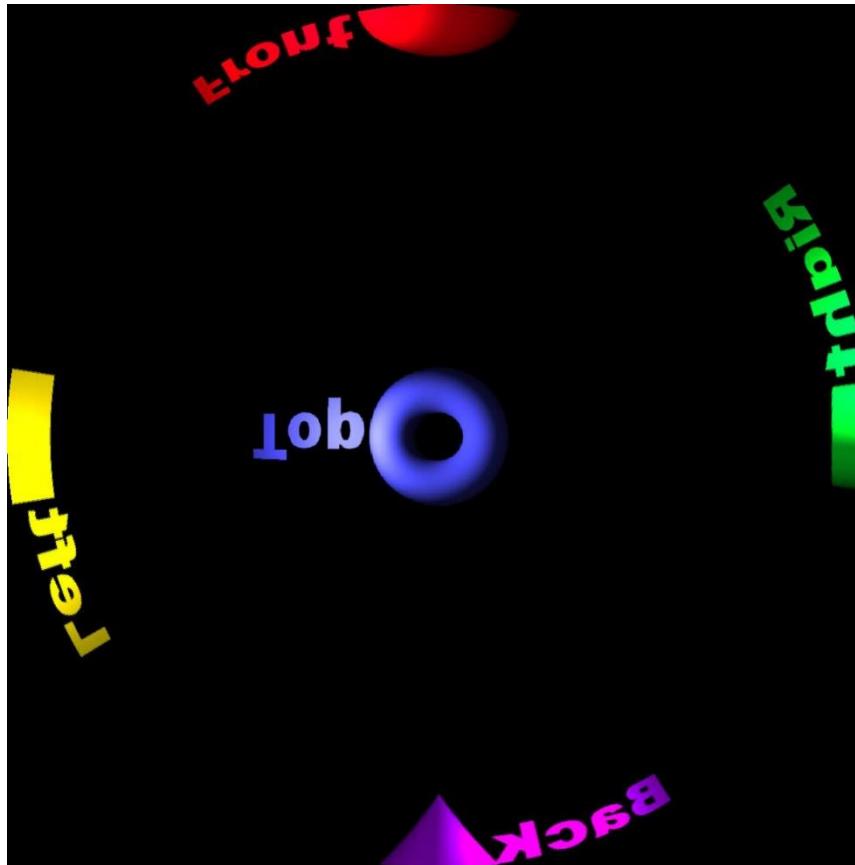
Fragment Shader

```
precision mediump float;  
  
varying vec3 R;  
uniform samplerCube texMap;  
  
void main()  
{  
    vec4 texColor = textureCube(texMap, R);  
    gl_FragColor = texColor;  
}
```

Sphere Mapping

- Original environmental mapping technique proposed by Blinn and Newell based in using lines of longitude and latitude to map parametric variables to texture coordinates
- OpenGL supports sphere mapping which requires a circular texture map equivalent to an image taken with a fisheye lens

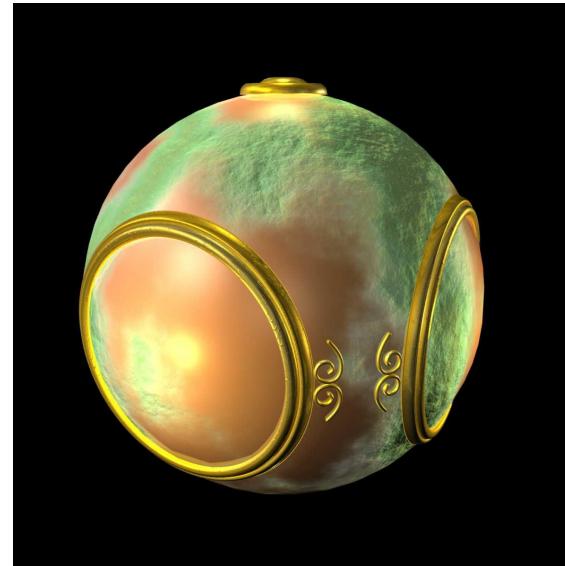
Sphere Map



Bump Maps

Objectives

- Introduce bump mapping

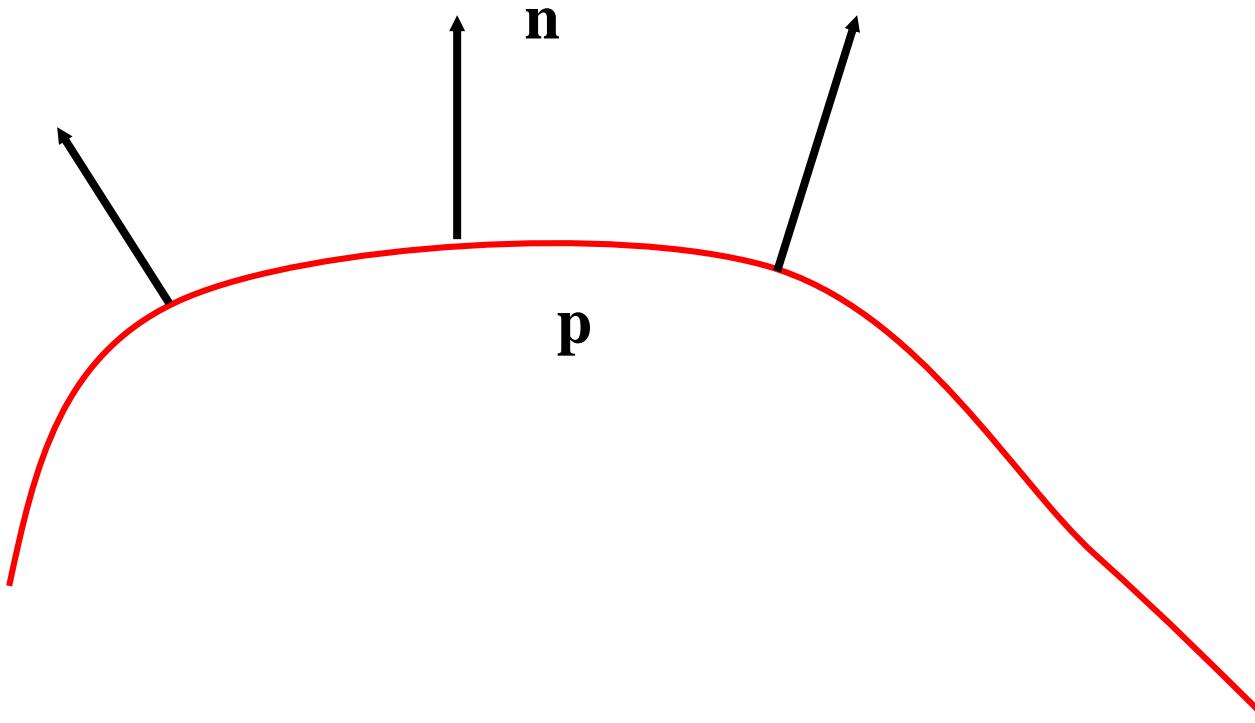


Modeling an Orange

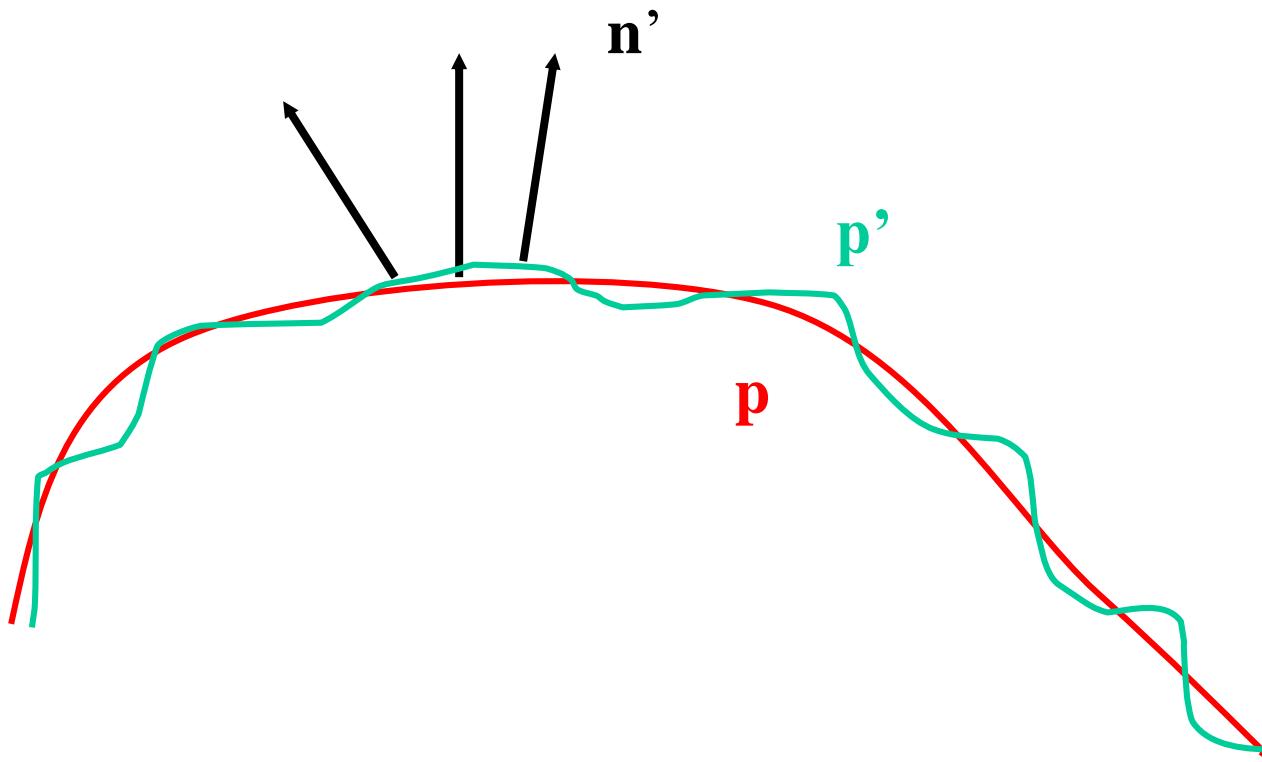
- Consider modeling an orange
- Texture map a photo of an orange onto a surface
 - Captures dimples
 - Will not be correct if we move viewer or light
 - We have shades of dimples rather than their correct orientation
- Ideally we need to **perturb normal across surface of object** and compute a new color at each interior point

Bump Mapping (Blinn)

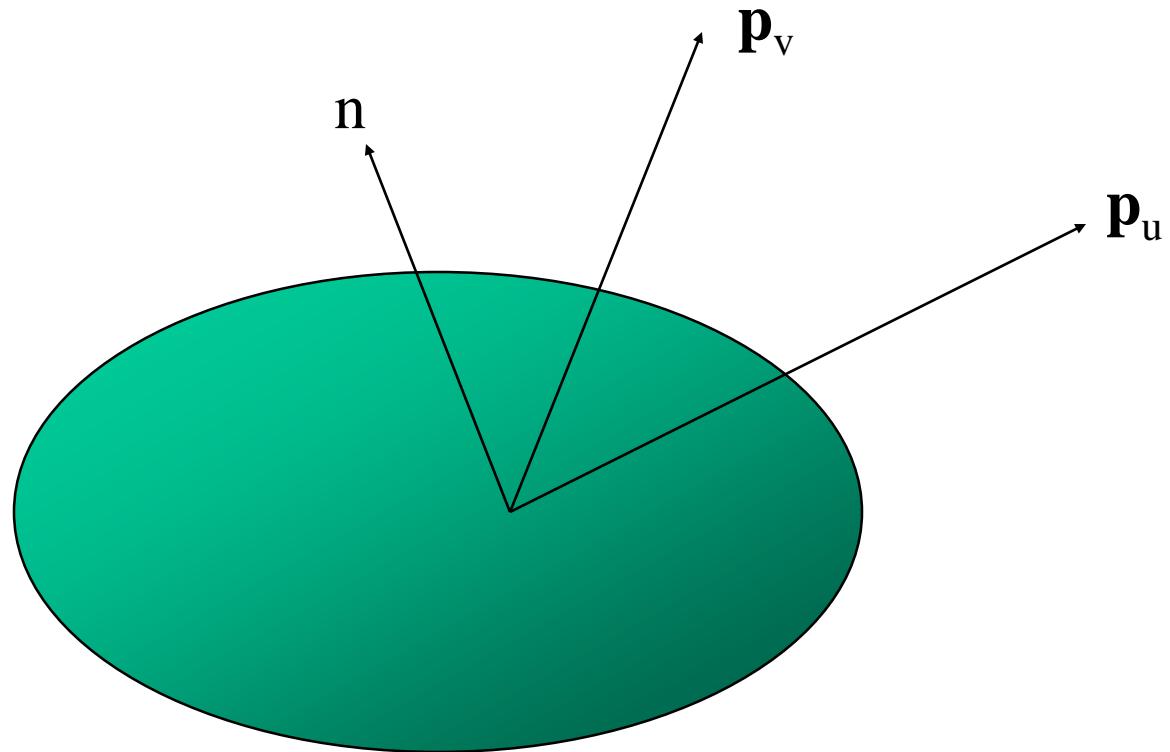
- Consider a smooth surface



Rougher Version



Tangent Plane



Equations

$$\mathbf{p}(u,v) = [x(u,v), y(u,v), z(u,v)]^T$$

$$\mathbf{p}_u = [\partial x / \partial u, \partial y / \partial u, \partial z / \partial u]^T$$

$$\mathbf{p}_v = [\partial x / \partial v, \partial y / \partial v, \partial z / \partial v]^T$$

$$\mathbf{n} = (\mathbf{p}_u \times \mathbf{p}_v) / \| \mathbf{p}_u \times \mathbf{p}_v \|$$

Displacement Function

$$\mathbf{p}' = \mathbf{p} + d(u,v) \mathbf{n}$$

$d(u,v)$ is the bump or displacement function

$$|d(u,v)| \ll 1$$

Perturbed Normal

$$\mathbf{n}' = \mathbf{p}'_u \times \mathbf{p}'_v$$

$$\mathbf{p}'_u = \mathbf{p}_u + (\partial d / \partial u) \mathbf{n} + d(u, v) \mathbf{n}_u$$

$$\mathbf{p}'_v = \mathbf{p}_v + (\partial d / \partial v) \mathbf{n} + d(u, v) \mathbf{n}_v$$

If d is small, we can neglect last term

Approximating the Normal

$$\mathbf{n}' = \mathbf{p}'_u \times \mathbf{p}'_v$$

$$\approx \mathbf{n} + (\partial d / \partial u) \mathbf{n} \times \mathbf{p}_v + (\partial d / \partial v) \mathbf{n} \times \mathbf{p}_u$$

The vectors $\mathbf{n} \times \mathbf{p}_v$ and $\mathbf{n} \times \mathbf{p}_u$ lie
in the tangent plane

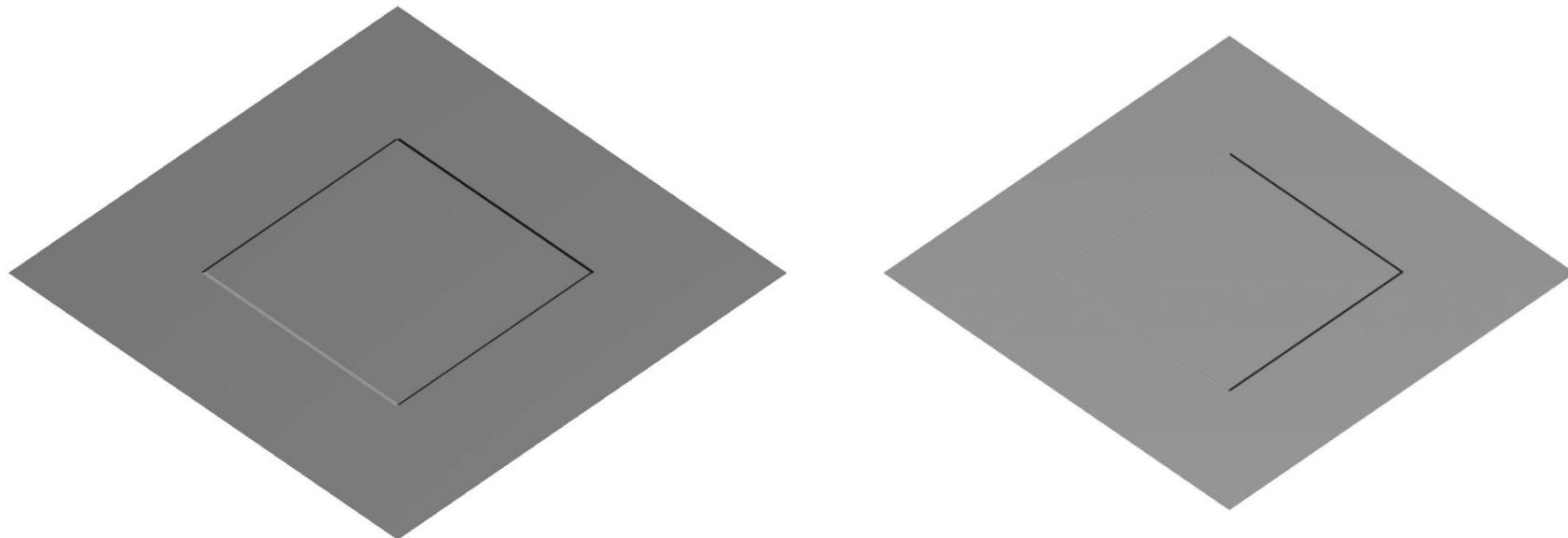
Hence the normal is displaced in the tangent plane
Must precompute the arrays $\partial d / \partial u$ and $\partial d / \partial v$
Finally, we perturb the normal during shading

Image Processing

- Suppose that we start with a function $d(u,v)$
- We can sample it to form an array $D=[d_{ij}]$
- Then $\partial d / \partial u \approx d_{ij} - d_{i-1,j}$
and $\partial d / \partial v \approx d_{ij} - d_{i,j-1}$
- **Embossing:** multipass approach using floating point buffer

Example

Single Polygon and a Rotating Light Source



How to do this?

- The problem is that we want to apply the perturbation at all points on the surface
- Cannot solve by vertex lighting (unless polygons are very small)
- Really want to apply to every fragment
- Can't do that in fixed function pipeline
- But can do with a fragment program!!
- See bumpmap.html and bumpmap.js

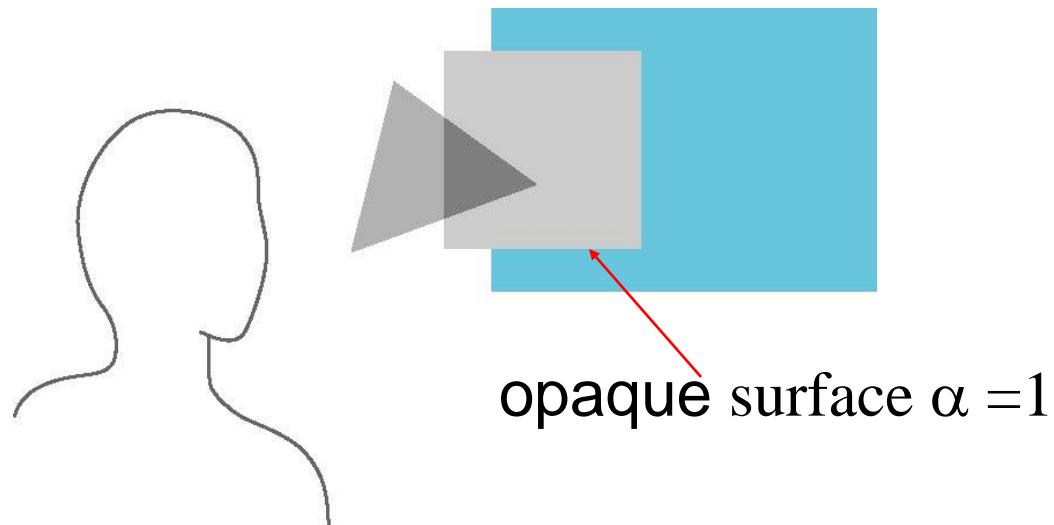
Compositing and Blending

Objectives

- Learn to use the A component in RGBA color for
 - Blending for translucent surfaces
 - Compositing images
 - Antialiasing

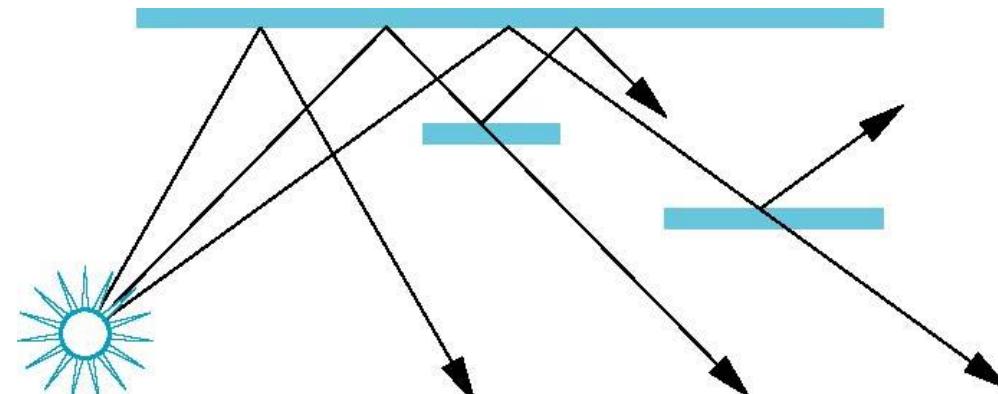
Opacity and Transparency

- **Opaque** surfaces permit no light to pass through
 - **Transparent** surfaces permit all light to pass
 - Translucent surfaces pass some light
- translucency** = $1 - \text{opacity} (\alpha)$



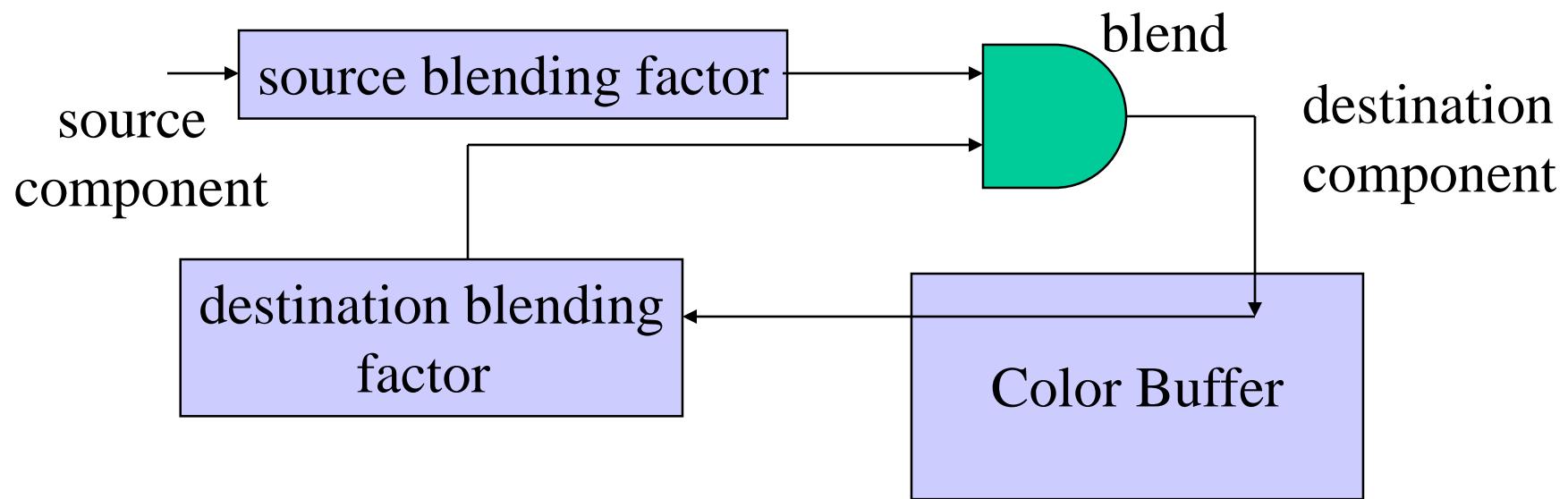
Physical Models

- Dealing with translucency in a physically correct manner is difficult due to
 - the complexity of the internal interactions of light and matter
 - Using a pipeline renderer



Writing Model

- Use A component of RGBA (or RGB α) color to store opacity
- During rendering we can expand our writing model to use RGBA values



Blending Equation

- We can define source and destination blending factors for each RGBA component

$$\mathbf{s} = [s_r, s_g, s_b, s_\alpha]$$

$$\mathbf{d} = [d_r, d_g, d_b, d_\alpha]$$

Suppose that the source and destination colors are

$$\mathbf{b} = [b_r, b_g, b_b, b_\alpha]$$

$$\mathbf{c} = [c_r, c_g, c_b, c_\alpha]$$

Blend as

$$\mathbf{c}' = [b_r s_r + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_\alpha s_\alpha + c_\alpha d_\alpha]$$

OpenGL Blending and Compositing

- Must enable blending and pick source and destination factors

```
gl.enable(gl.BLEND)
```

```
gl.blendFunc(source_factor, destination_factor)
```

- Only certain factors supported
 - `gl.ZERO`, `gl.ONE`
 - `gl.SRC_ALPHA`, `gl.ONE_MINUS_SRC_ALPHA`
 - `gl.DST_ALPHA`, `gl.ONE_MINUS_DST_ALPHA`
 - See Redbook for complete list

Example

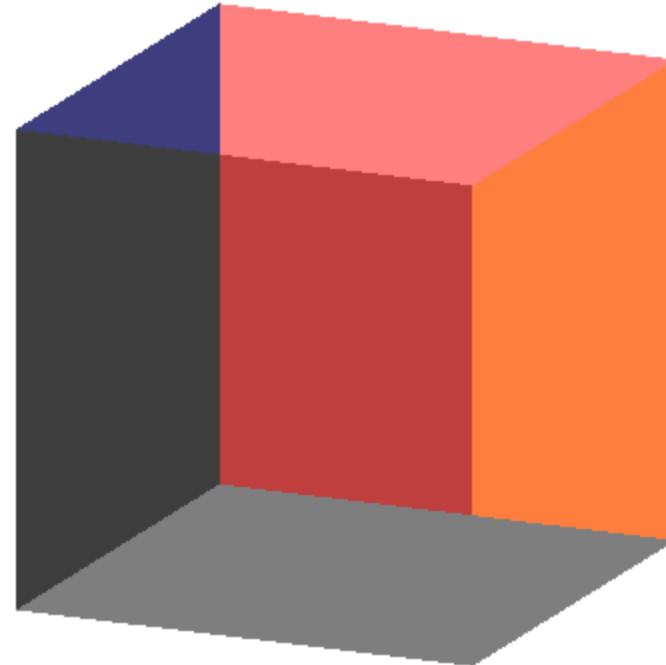
- Suppose that we start with the opaque background color $(R_0, G_0, B_0, 1)$
 - This color becomes the initial destination color
- We now want to blend in a translucent polygon with color $(R_1, G_1, B_1, \alpha_1)$
- Select `GL_SRC_ALPHA` and `GL_ONE_MINUS_SRC_ALPHA` as the source and destination blending factors
$$R' = \alpha_1 R_1 + (1 - \alpha_1) R_0, \dots$$
- Note this formula is correct if polygon is either opaque or transparent

Clamping and Accuracy

- All the components (RGBA) are **clamped** and stay in the range (0,1)
- However, in a typical system, RGBA values are only stored to 8 bits
 - Can **easily loose accuracy** if we add many components together
 - Example: add together n images
 - Divide all color components by n to avoid clamping
 - Blend with source factor = 1, destination factor = 1
 - But division by n loses bits

Order Dependency

- Is this image correct?
 - Probably not
 - Polygons are rendered in the order they pass down the pipeline
 - Blending functions are **order dependent**



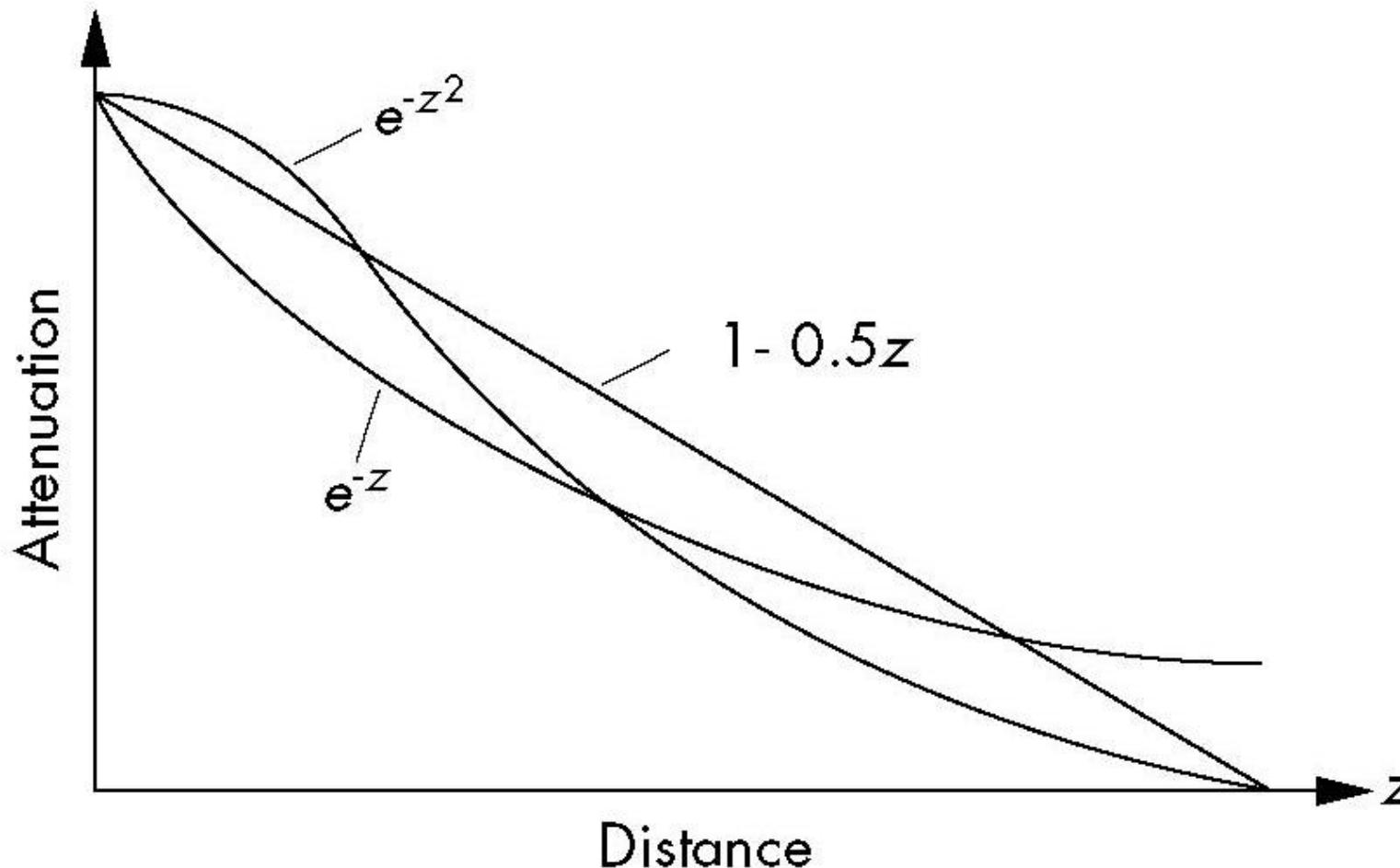
Opaque and Translucent Polygons

- Suppose that we have a group of polygons some of which are opaque and some translucent
- How do we use hidden-surface removal?
- **Opaque polygons** block all polygons behind them and affect the depth buffer
- Translucent polygons should not affect depth buffer
 - Render with `gl.depthMask(false)` which makes depth buffer read-only
- Sort polygons first to remove order dependency

Fog

- We can composite with a fixed color and have the blending factors depend on depth
 - Simulates a fog effect
- Blend source color C_s and fog color C_f by
$$C'_s = f C_s + (1-f) C_f$$
- f is the *fog factor*
 - Exponential
 - Gaussian
 - Linear (depth cueing)
- Deprecated but can recreate

Fog Functions

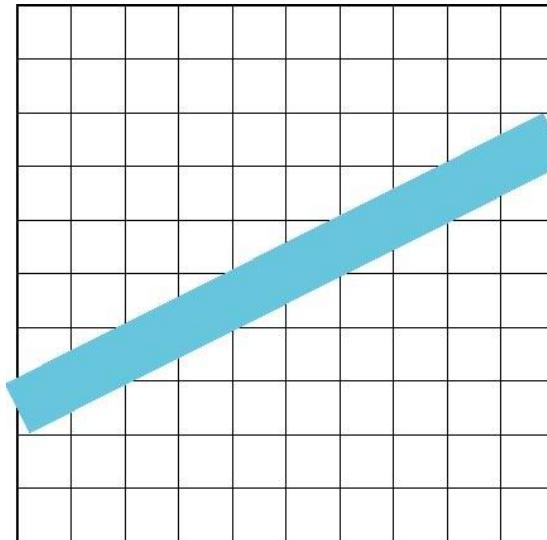


Compositing and HTML

- In desktop OpenGL, the A component has no effect unless blending is enabled
- In WebGL, an A other than 1.0 has an effect because WebGL works with the HTML5 Canvas element
- $A = 0.5$ will cut the RGB values by $\frac{1}{2}$ when the pixel is displayed
- Allows other applications to be blended into the canvas along with the graphics

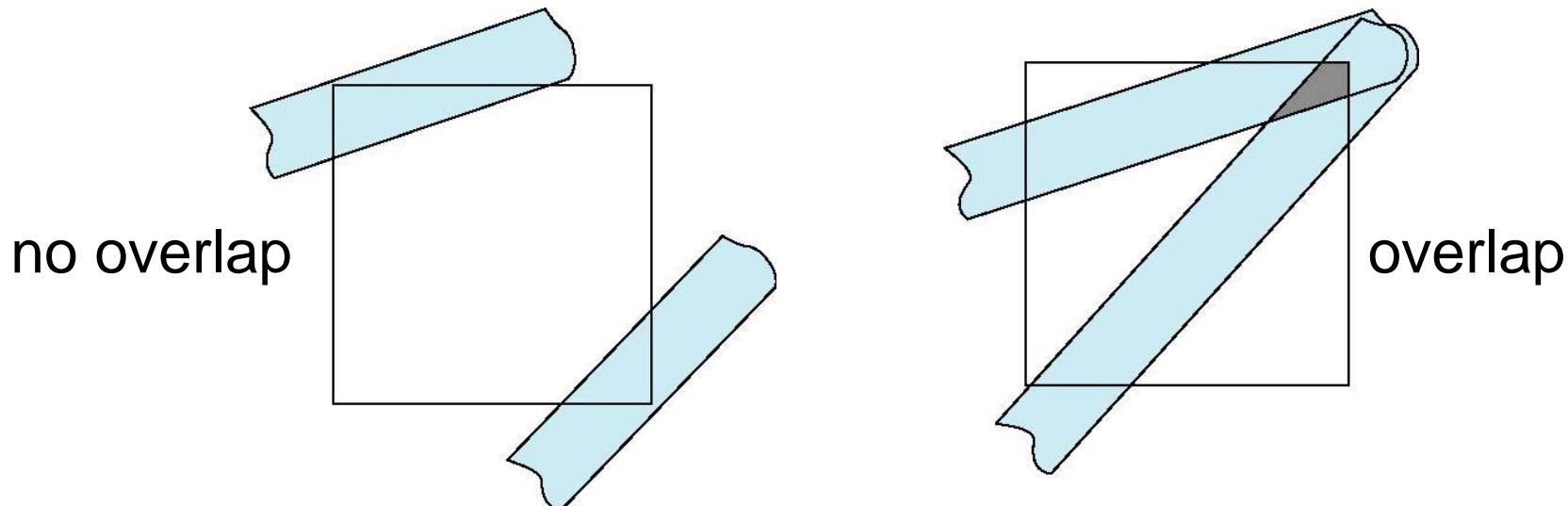
Line Aliasing

- Ideal raster line is one pixel wide
- All line segments, other than vertical and horizontal segments, partially cover pixels
- Simple algorithms color only whole pixels
- Lead to the “jaggies” or aliasing
- Similar issue for polygons



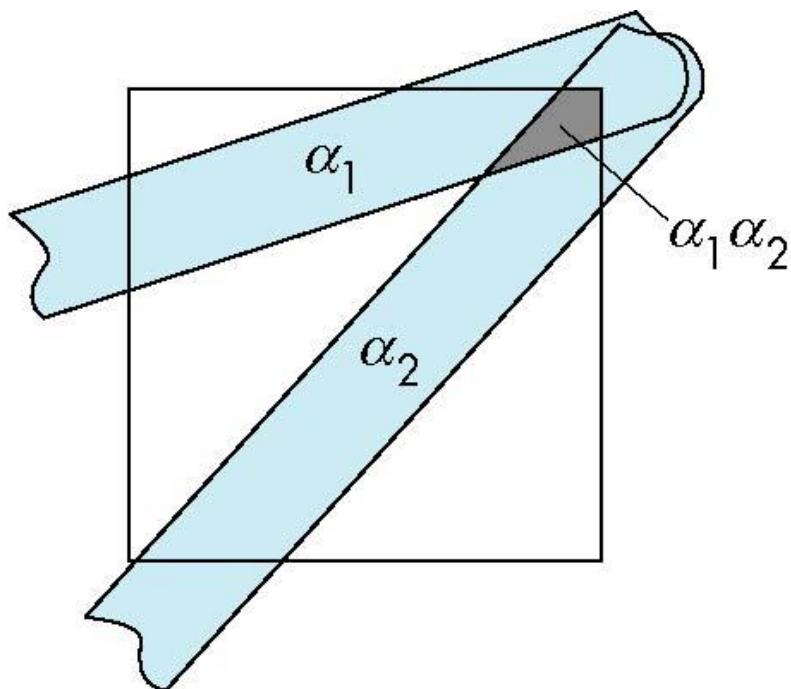
Antialiasing

- Can try to color a pixel by adding a fraction of its color to the frame buffer
 - Fraction depends on percentage of pixel covered by fragment
 - Fraction depends on whether there is overlap



Area Averaging

- Use average area $\alpha_1 + \alpha_2 - \alpha_1\alpha_2$ as blending factor



OpenGL Antialiasing

- Not (yet) supported in WebGL
- Can enable separately for points, lines, or polygons

```
glEnable(GL_POINT_SMOOTH) ;  
glEnable(GL_LINE_SMOOTH) ;  
glEnable(GL_POLYGON_SMOOTH) ;  
  
glEnable(GL_BLEND) ;  
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA) ;
```

- Note most hardware will automatically antialias

Imaging Applications

Objectives

- Use the fragment shader to do image processing
 - Image filtering
 - Pseudo Color
- Use multiple textures
 - matrix operations
- Introduce GPGPU

Accumulation Techniques

- Compositing and blending are limited by resolution of the frame buffer
 - Typically 8 bits per color component
- The *accumulation buffer* was a high resolution buffer (16 or more bits per component) that avoided this problem
- Could write into it or read from it with a scale factor
- Slower than direct compositing into the frame buffer
- Now deprecated but can do techniques with floating point frame buffers

Multirendering

- Composite multiple images
- Image Filtering (convolution)
 - add shifted and scaled versions of an image
- Whole scene antialiasing
 - move primitives a little for each render
- Depth of Field
 - move viewer a little for each render keeping one plane unchanged
- Motion effects

Fragment Shaders and Images

- Suppose that we send a rectangle (two triangles) to the vertex shader and render it with an $n \times m$ texture map
- Suppose that in addition we use an $n \times m$ canvas
- There is now a one-to-one correspondence between each texel and each fragment
- Hence we can **regard fragment operations as imaging operations on the texture map**

GPGPU

- Looking back at these examples, we can note that the only purpose of the geometry is to trigger the execution of the imaging operations in the fragment shader
- Consequently, we can look at what we have done as large matrix operations rather than graphics operations
- Leads to the field of General Purpose Computing with a GPU (GPGPU)

Examples

- Add two matrices
- Multiply two matrices
- Fast Fourier Transform
- Uses speed and parallelism of GPU
- But how do we get out results?
 - Floating point frame buffers
 - OpenCL (WebCL)
 - Compute shaders

Using Multiple Texels

- Suppose we have a 1024 x 1024 texture in the texture object “image”

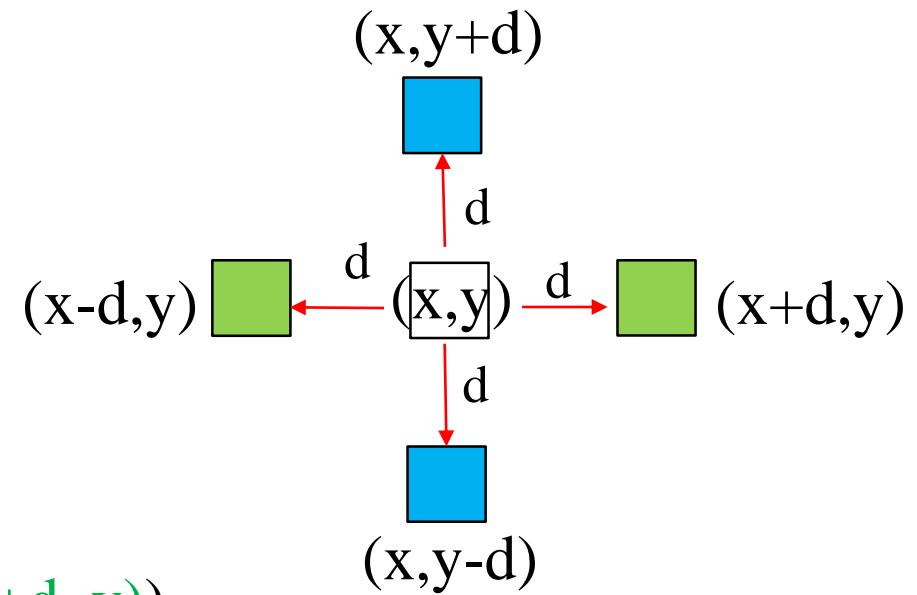
`sampler2D(image, vec2(x,y))` returns the value of the texture at `(x,y)`

`sampler2D(image, vec2(x+1.0/1024.0, y))`; returns the value of the texel to `the right of (x,y)`

We can use any combination of texels surrounding (x, y) in the fragment shader

Image Enhancer

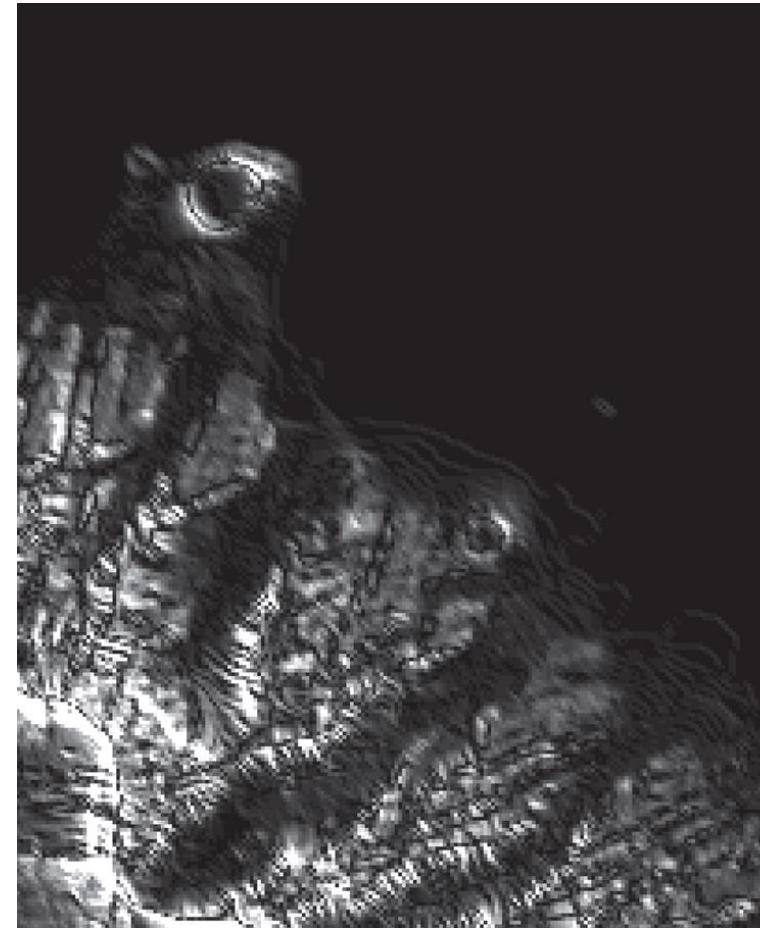
```
precision mediump float;  
varying vec2 fTexCoord;  
uniform sampler2D texture;  
void main()  
{  
    float d = 1.0/256.0; //spacing between texels  
    float x = fTexCoord.x;  
    float y = fTexCoord.y;  
  
    gl_FragColor = 10.0*abs( texture2D( texture, vec2(x+d, y))  
                            - texture2D( texture, vec2(x-d, y)))  
                  +10.0*abs( texture2D( texture, vec2(x, y+d))  
                            - texture2D( texture, vec2(x, y-d)));  
  
    gl_FragColor.w = 1.0;  
}
```



Honolulu Image



original



enhanced

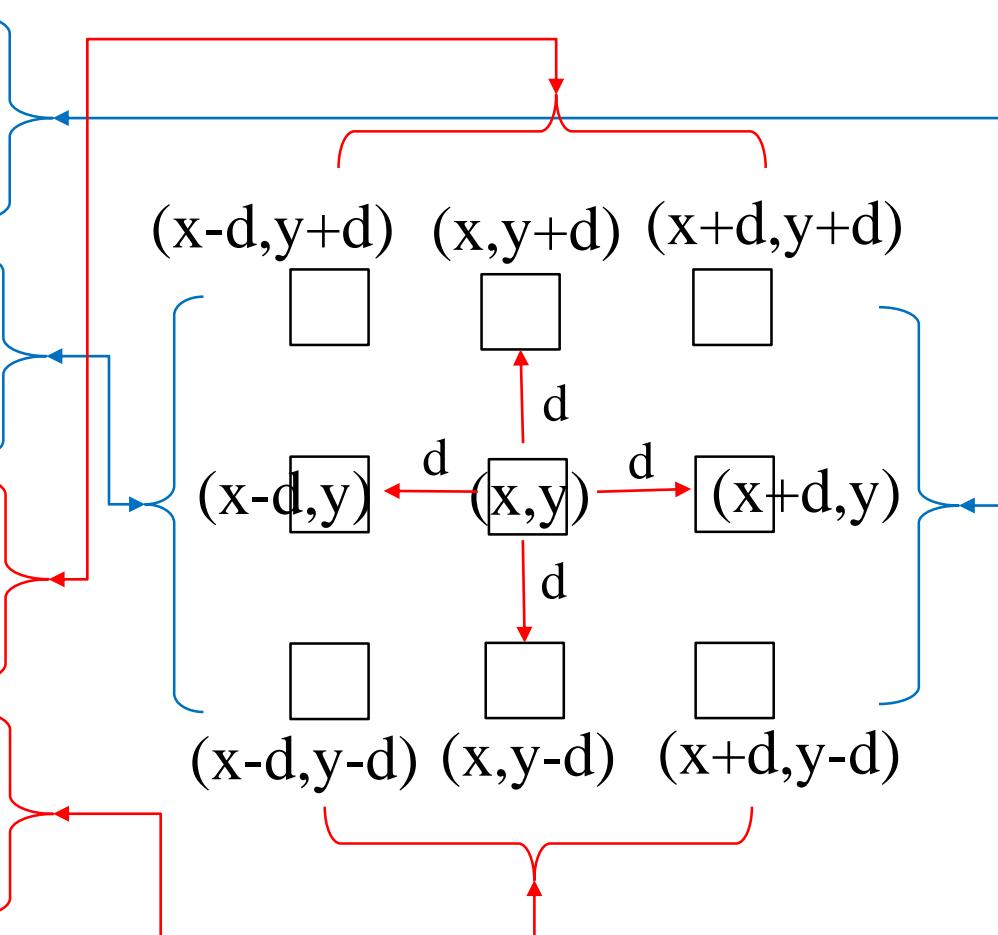
Sobel Edge Detector

- Nonlinear
- Find approximate gradient at each point
- Compute smoothed finite difference approximations to x and y components separately
- Display magnitude of approximate gradient
- Simple with fragment shader

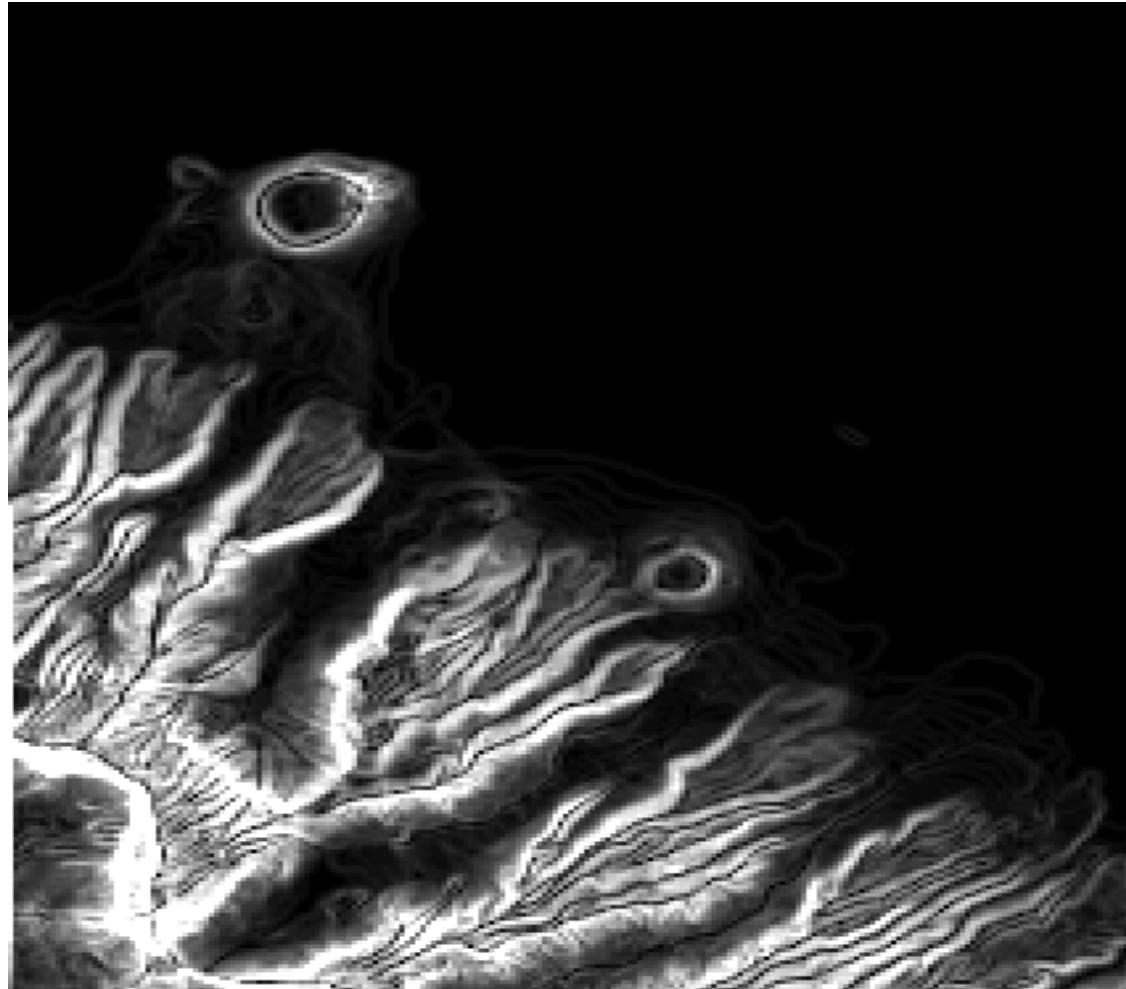
Sobel Edge Detector

```
vec4 gx = 3.0*texture2D( texture, vec2(x+d, y))  
    + texture2D( texture, vec2(x+d, y+d))  
    + texture2D( texture, vec2(x+d, y-d))  
- 3.0*texture2D( texture, vec2(x-d, y))  
- texture2D( texture, vec2(x-d, y+d))  
- texture2D( texture, vec2(x-d, y-d));  
  
vec4 gy = 3.0*texture2D( texture, vec2(x, y+d))  
    + texture2D( texture, vec2(x+d, y+d))  
    + texture2D( texture, vec2(x-d, y+d))  
- 3.0*texture2D( texture, vec2(x, y-d))  
- texture2D( texture, vec2(x+d, y-d))  
- texture2D( texture, vec2(x-d, y-d));
```

```
gl_FragColor = vec4(sqrt(gx*gx + gy*gy), 1.0);  
gl_FragColor.w = 1.0;
```



Sobel Edge Detector



Using Multiple Textures

- Example: matrix addition
- Create two samplers, texture1 and texture2, that contain the data
- In fragment shader

```
gl_FragColor =  
    sampler2D(texture1, vec2(x, y)) +sampler2D(texture2,  
    vec2(x,y));
```

Using 4 Way Parallelism

- Recent GPUs and graphics cards support textures up to 8K x 8K
- For scalar imaging, we can do twice as well using all four color components

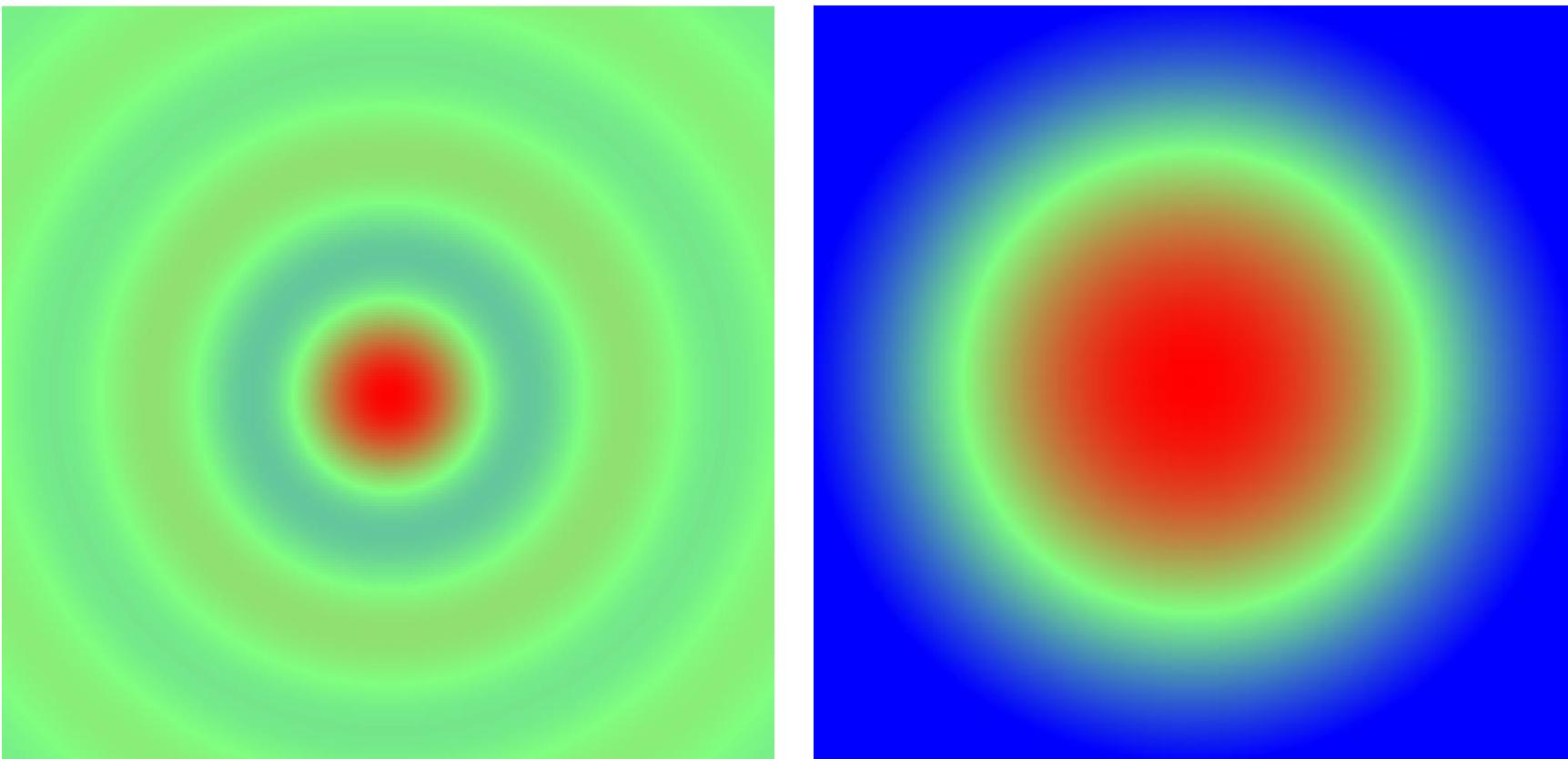
$$\begin{bmatrix} R & G \\ B & A \end{bmatrix}$$

Indexed and Pseudo Color

- Display luminance (2D) image as texture map
- Treat pixel value as independent variable for separate functions for each color component

```
void main(){  
    vec4 color = texture2D(texture, fTexCoord);  
    if(color.g<0.5) color.g = 2.0*color.g;  
        else color.g = 2.0 - 2.0*color.g;  
    color.b = 1.0-color.b;  
    gl_FragColor = color;  
}
```

Top View of 2D Sinc



The Next Step

- Need more storage for most GPGPU calculations
- Example: filtering
- Example: iteration
- Need shared memory
- Solution: Use texture memory and off-screen rendering

Framebuffer Objects

Objectives

- Look at methods that use memory on the graphics card
- Introduce off screen rendering
- Learn how to create framebuffer objects
 - Create a renderbuffer
 - Attach resources

Discrete Processing in WebGL

- Recent GPUs contain large amounts of memory
 - Texture memory
 - Framebuffer
 - Floating point
- Fragment shaders support discrete operations at the pixel level
- Separate pixel (texel) pipeline

Accessing the Framebuffer

- Pre 3.1 OpenGL had functions that allowed access to the framebuffer and other OpenGL buffers
 - Draw Pixels
 - Read Pixels
 - Copy Pixels
 - BitBlt
 - Accumulation Buffer functions
- All deprecated

Going between CPU and GPU

- We have already seen that we can write pixels as texels to texture memory
- Texture objects reduce transfers between CPU and GPU
- Transfer of pixel data back to CPU slow
- Want to manipulate pixels without going back to CPU
 - Image processing
 - GPGPU

Framebuffer Objects

- Framebuffer Objects (FBOs) are buffers that are created by the application
 - Not under control of window system
 - Cannot be displayed
 - Can attach a **renderbuffer** to a FBO and can **render off screen** into the attached buffer
 - Attached buffer can then be detached and used as a texture map for an on-screen render to the default frame buffer

Render to Texture

- Textures are shared by all instances of the fragment shader
- If we render to a texture attachment we can create a new texture image that can be used in subsequent renderings
- Use a double buffering strategy for operations such as convolution

Steps

- Create an Empty Texture Object
- Create a FBO
- Attach renderbuffer for texture image
- Bind FBO
- Render scene
- Detach renderbuffer
- Bind texture
- Render with new texture

Specify a two-dimensional texture image

Empty Texture Object

```
texture1 = gl.createTexture();
gl.activeTexture( gl.TEXTURE0 );
gl.bindTexture( gl.TEXTURE_2D, texture1 );
```

```
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 512, 512, 0, gl.RGBA, gl.UNSIGNED_BYTE, null );
```

```
gl.generateMipmap(gl.TEXTURE_2D);
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST )
```

R, G, B, and A components are read from the color buffer

Specify the width/height of the texture

Specify the width of the border. Must be 0.

The format of the texel data

The data type of the texel data

A Unit8array: a pixel source for the texture

Specify the level of detail. Level 0 is the base image level and level n is the n th mipmap reduction level.

Creating a FBO

- We create a framebuffer object in a similar manner to other objects
- Creating an FBO creates an empty FBO
- Must add needed resources
 - Can add **a renderbuffer to render into**
 - Can add **a texture** which can also be rendered into
 - For **hidden surface removal** we must add **a depth buffer attachment** to the renderbuffer

Attach a texture
to a framebuffer

Frame Buffer Object

Collection buffer data
storage of color, alpha,
depth and stencil
buffers used to render
an image.

```
var framebuffer = gl.createFramebuffer();  
gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);  
framebuffer.width = 512;  
framebuffer.height = 512;
```

Attach the texture to
the framebuffer's color
buffer

```
//renderbuffer = gl.createRenderbuffer();  
//gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);  
//gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16, 512, 512);  
// Attach color buffer
```

Specify the texture
target: a 2D image

```
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture1, 0);  
//gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.RENDERBUFFER, renderbuffer);  
// check for completeness  
var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);  
if(status != gl.FRAMEBUFFER_COMPLETE) alert('Frame Buffer Not Complete');
```

An object whose
image to attach

Specify the mipmap level
of the texture image to be
attached. Must be 0.

Rest of Initialization

- Same as previous examples
 - Allocate VAO
 - Fill VAO with data for render to texture
- Initialize two program objects with different shaders
 - First for render to texture
 - Second for rendering with created texture

Render to Texture

Objectives

- Examples of render-to-texture
- Render a triangle to texture, then use this texture on a rectangle
- Introduce buffer pingponging

Program Objects and Shaders

- For most applications of render-to-texture we need multiple program objects and shaders
 - One set for creating a texture
 - Second set for rendering with that texture
- Applications that we consider later such as buffer pingponging may require additional program objects

Program Object 1 Shaders

pass through **vertex shader**:

```
attribute vec4 vPosition;  
void main()  
{  
    gl_Position = vPosition;  
}
```

fragment shader to get a red triangle:

```
precision mediump float;  
void main()  
{  
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);  
}
```

Program Object 2 Shaders

```
// vertex shader  
  
attribute vec4 vPosition;  
attribute vec2 vTexCoord;  
varying vec2 fTexCoord;  
void main()  
{  
    gl_Position = vPosition;  
    fTexCoord = vTexCoord;  
}
```

```
// fragment shader  
  
precision mediump float;  
  
varying vec2 fTexCoord;  
uniform sampler2D texture;  
void main()  
{  
    gl_FragColor = texture2D( texture, fTexCoord );  
}
```

First Render (to Texture)

```
gl.useProgram( program1 );
var buffer1 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer1 );
gl.bufferData( gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW );

// Initialize the vertex position attribute from the vertex shader

var vPosition = gl.getAttribLocation( program1, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );

// Render one triangle

gl.viewport(0, 0, 64, 64);
gl.clearColor(0.5, 0.5, 0.5, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT );
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Set Up Second Render

```
// Bind to default window system framebuffer
```

```
gl.bindFramebuffer(gl.FRAMEBUFFER, null);  
gl.disableVertexAttribArray(vPosition);  
gl.useProgram(program2);
```

```
// Assume we have already set up a texture object with null texture image
```

```
gl.activeTexture(gl.TEXTURE0);  
gl.bindTexture(gl.TEXTURE_2D, texture1);
```

```
// set up vertex attribute arrays for texture coordinates and rectangle as usual
```

Specify which texture unit to make active.

Bind a given texture to a target (binding point).

Data for Second Render

```
var buffer2 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer2);
gl.bufferData(gl.ARRAY_BUFFER, new flatten(vertices), gl.STATIC_DRAW);

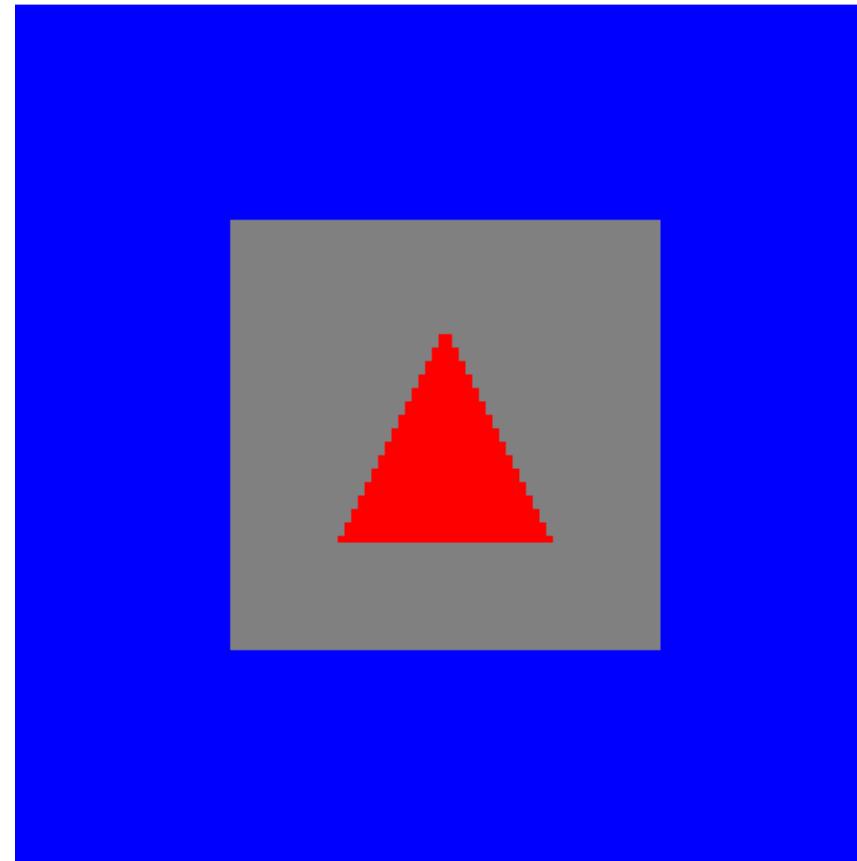
var vPosition = gl.getAttribLocation( program2, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );

var buffer3 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer3);
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoord), gl.STATIC_DRAW);

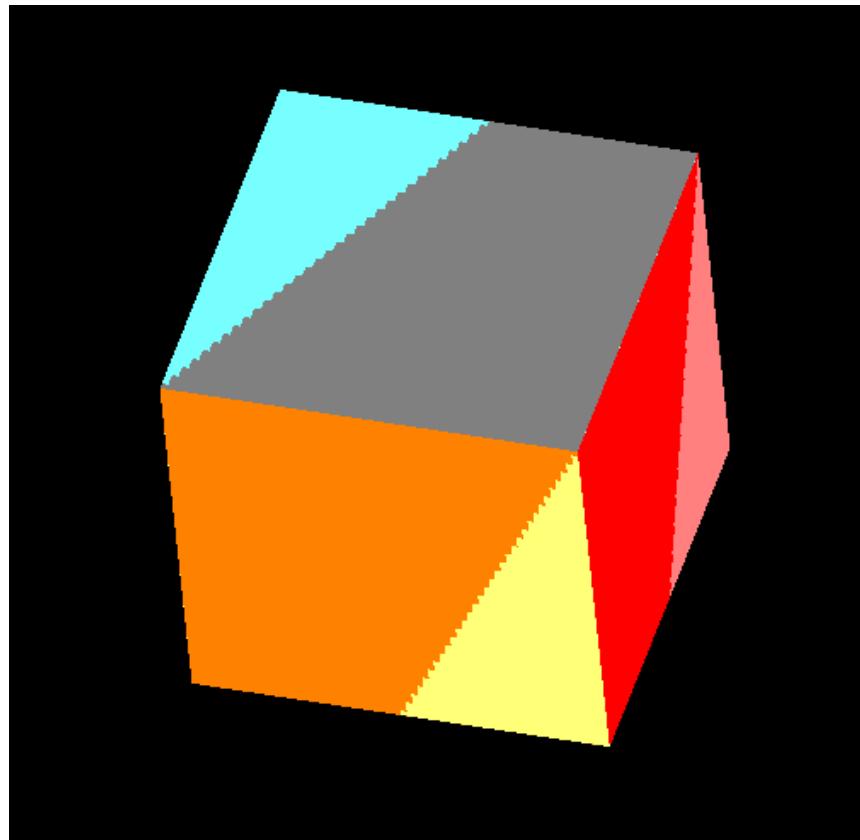
var vTexCoord = gl.getAttribLocation( program2, "vTexCoord" );
gl.vertexAttribPointer( vTexCoord, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vTexCoord );
```

Render a Quad with Texture

```
gl.uniform1i( gl.getUniformLocation(program2, "texture"), 0);  
  
gl.viewport(0, 0, 512, 512);  
gl.clearColor(0.0, 0.0, 1.0, 1.0);  
gl.clear( gl.COLOR_BUFFER_BIT );  
  
gl.drawArrays(gl.TRIANGLES, 0, 6);
```



Dynamic 3D Example



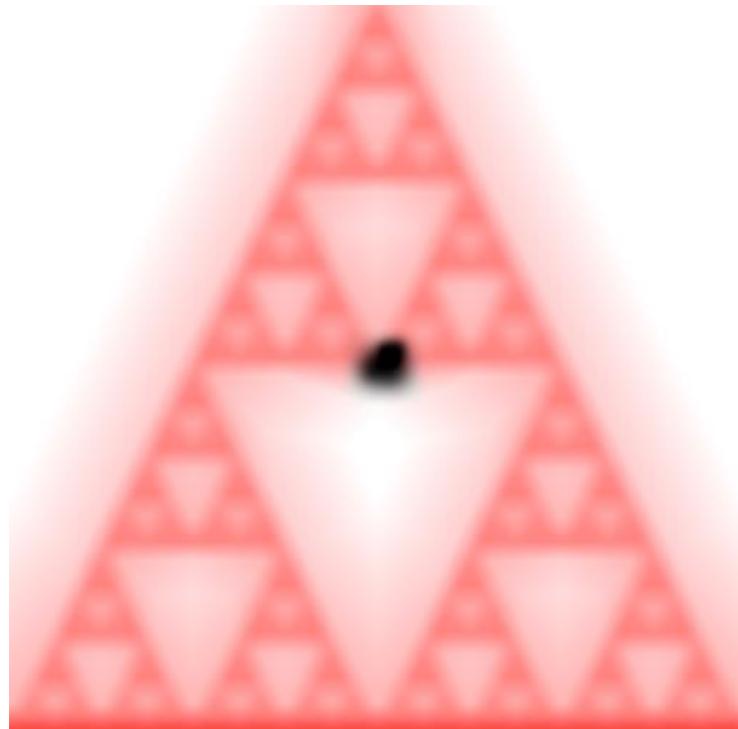
Buffer Ping-ponging

- Iterative calculations can be accomplished using multiple render buffers
- Original data in texture buffer 1
- Render to texture buffer 2
- Swap buffers and rerender to texture

Particle System Example

- Random motion of a particle
 - Render as a point
 - Diffuse rendered image to create motion blur effect
 - Insert particle again in new position
- Example use Sierpinski gasket as initial background
- Uses three program objects

Screen Shots



Picking by Color

Objectives

- Use off-screen rendering for picking
- Example: rotating cube with shading
 - indicate which face is clicked on with mouse
 - normal rendering uses vertex colors that are interpolated across each face
 - Vertex colors could be determined by lighting calculation or just assigned
 - use console log to indicate which face (or background) was clicked

Algorithm

- Assign a unique color to each object
- When the mouse is clicked:
 - Do an off-screen render using these colors and no lighting
 - use gl.readPixels to obtain the color of the pixel where the mouse is located
 - map the color to the object id
 - do a normal render to the display

Shaders

- Only need one program object
- Vertex shader: same as in previous cube examples
 - includes rotation matrices
 - gets angle as uniform variable
- Fragment shader
 - Stores face colors for picking
 - Gets vertex color for normal render from rasterizer
- Send uniform integer to fragment shader as index for desired color

Fragment Shader

```
precision mediump float;  
  
uniform int i;  
varying vec4 fColor;  
void main()  
{  
    vec4 c[7];  
    c[0] = fColor;  
    c[1] = vec4(1.0, 0.0, 0.0, 1.0);  
    c[2] = vec4(0.0, 1.0, 0.0, 1.0);  
    c[3] = vec4(0.0, 0.0, 1.0, 1.0);  
    c[4] = vec4(1.0, 1.0, 0.0, 1.0);  
    c[5] = vec4(0.0, 1.0, 1.0, 1.0);  
    c[6] = vec4(1.0, 0.0, 1.0, 1.0);
```

Fragment Shader

```
// no case statement in GLSL  
  
if(i==0) gl_FragColor = c[0];  
else if(i==1) gl_FragColor = c[1];  
else if(i==2) gl_FragColor = c[2];  
else if(i==3) gl_FragColor = c[3];  
else if(i==4) gl_FragColor = c[4];  
else if(i==5) gl_FragColor = c[5];  
else if(i==6) gl_FragColor = c[6];  
}
```

Setup

```
// Allocate a frame buffer object
```

```
framebuffer = gl.createFramebuffer();  
gl.bindFramebuffer( gl.FRAMEBUFFER, framebuffer);
```

```
// Attach color buffer
```

```
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,  
                        gl.TEXTURE_2D, texture, 0);  
gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

the texture target:
a 2D image

A texture object whose
image to attach

Specify the mipmap level of the texture
image to be attached. Must be 0.

Attach a texture
to a framebuffer

A framebuffer
used to render
an image

Attach the
texture to the
framebuffer's
color buffer

Event Listener

```
canvas.addEventListener("mousedown", function(){
    gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
    gl.clear( gl.COLOR_BUFFER_BIT);
    gl.uniform3fv(thetaLoc, theta);
    for(var i=0; i<6; i++) {
        gl.uniform1i(gl.getUniformLocation(program, "i"), i+1);
        gl.drawArrays( gl.TRIANGLES, 6*i, 6 );
    }
    var x = event.clientX;
    var y = canvas.height -event.clientY;
    gl.readPixels(x, y, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, color);
});
```

the format of the pixel data

the data type of the pixel data

An array object to read data into

(x,y):
the horizontal/vertically
pixel that is read from the
lower left corner of a
rectangular block of pixels.

(1,1): the width/height of
the rectangle

1, 1

the data type of the pixel data

An array object to read data into

Event Listener

```
if(color[0]==255)
    if(color[1]==255) console.log("yellow");
    else if(color[2]==255) console.log("magenta");
    else console.log("red");

else if(color[1]==255)
    if(color[2]==255) console.log("cyan");
    else console.log("green");

else if(color[2]==255) console.log("blue");
else console.log("background");
```

Event Listener

```
// return to default framebuffer  
    gl.bindFramebuffer(gl.FRAMEBUFFER, null);  
  
//send index 0 to fragment shader  
    gl.uniform1i(gl.getUniformLocation(program, "i"), 0);  
  
//normal render  
    gl.clear( gl.COLOR_BUFFER_BIT );  
    gl.uniform3fv(thetaLoc, theta);  
    gl.drawArrays(gl.TRIANGLES, 0, 36);  
});
```

Picking by Selection

- Possible with render-to-texture
- When mouse clicked do a off screen rendering with new viewing conditions that render only a small area around mouse
- Keep track of what gets rendered to this off screen buffer

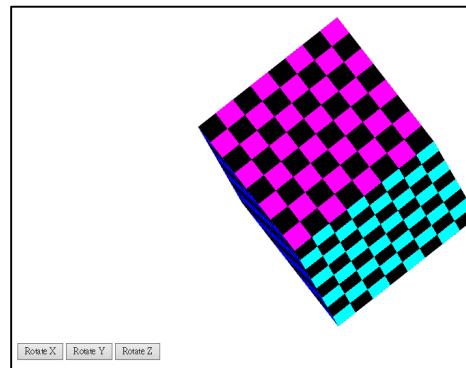
Sample Programs

Sample Programs

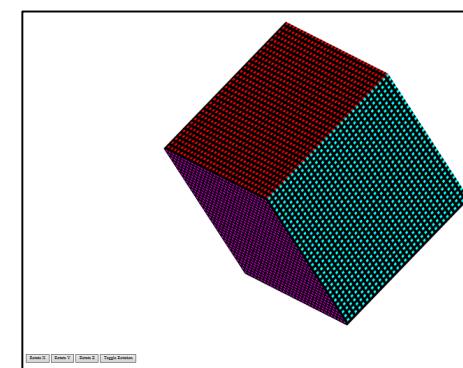
textureCube1



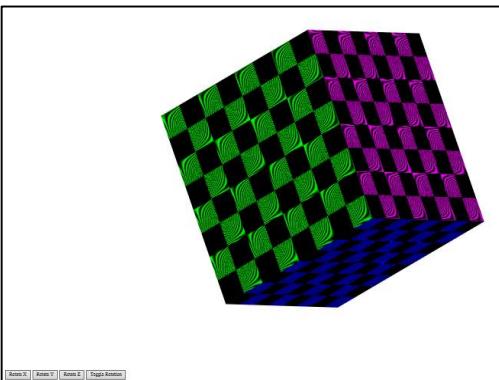
textureCubev2



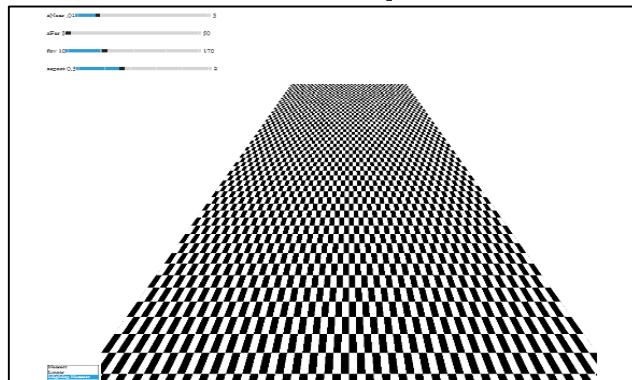
textureCubev3



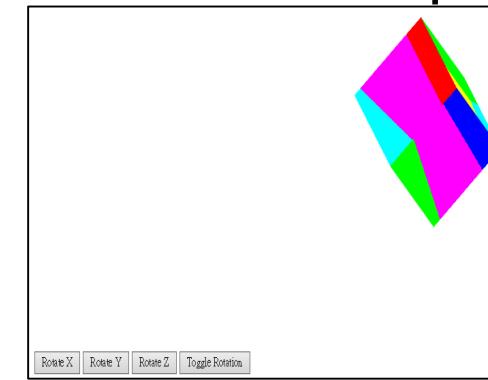
textureCube4



textureSquare

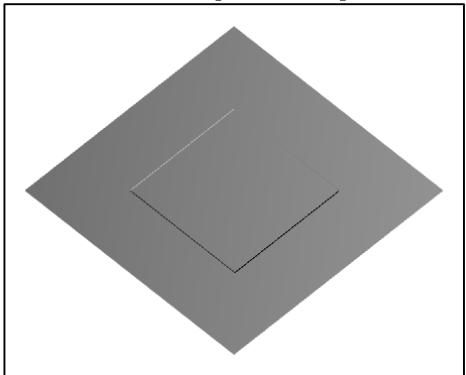


reflectionMap

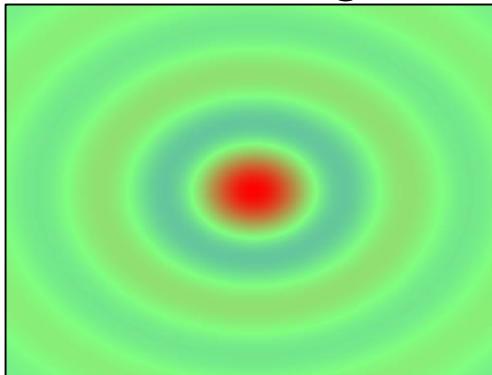


Sample Programs

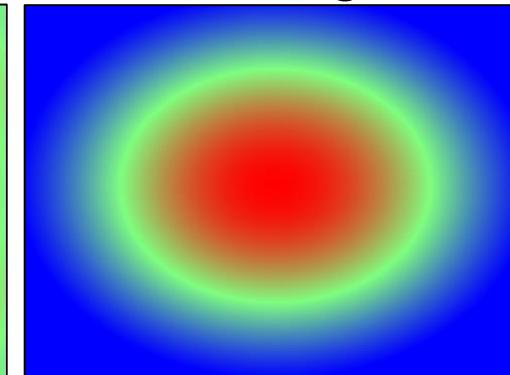
bumpMap



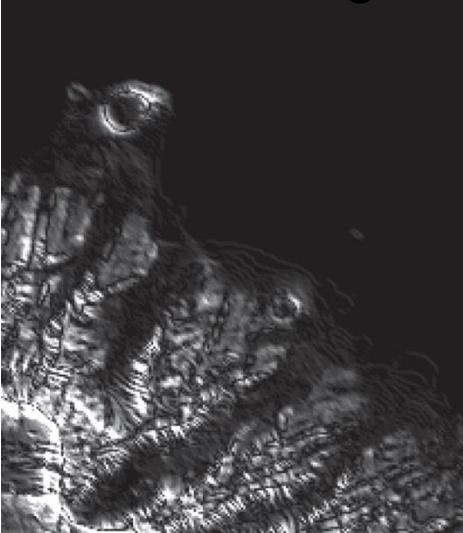
hatImage



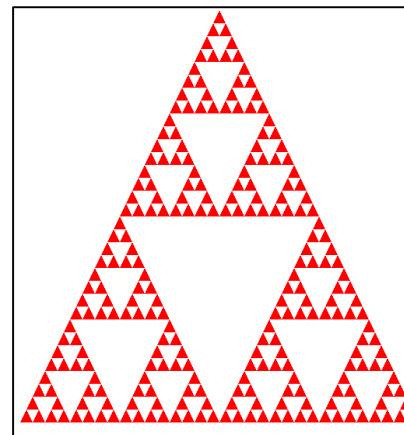
hatImage2



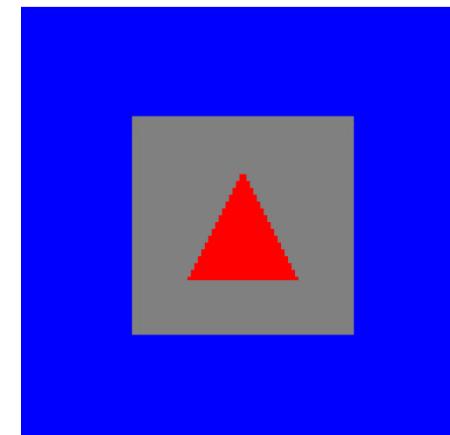
honoluluImage



render1

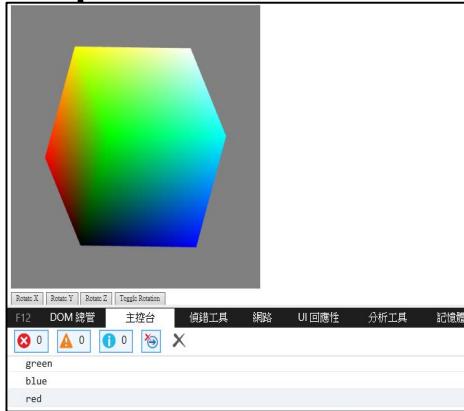


render1v2

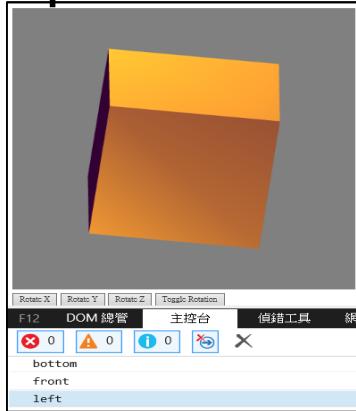


Sample Programs

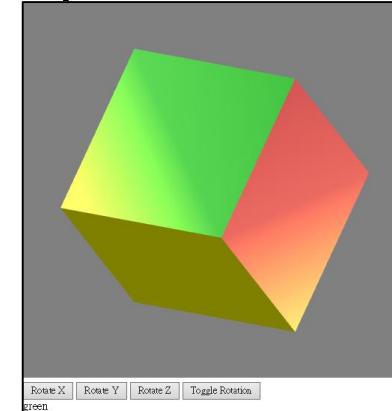
pickCube



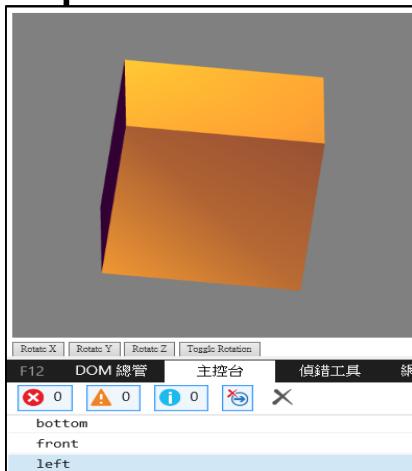
pickCube2



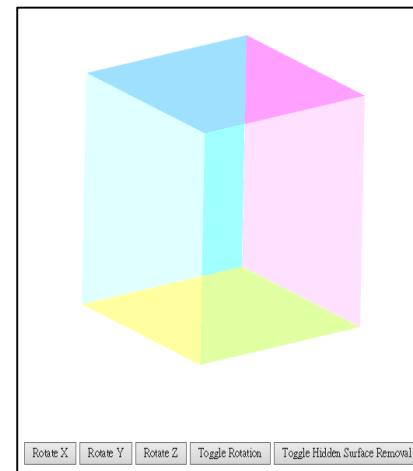
pickCube3



pickCube4



cubet



Sample Programs: `textureCube1.html`, `textureCube1.js`

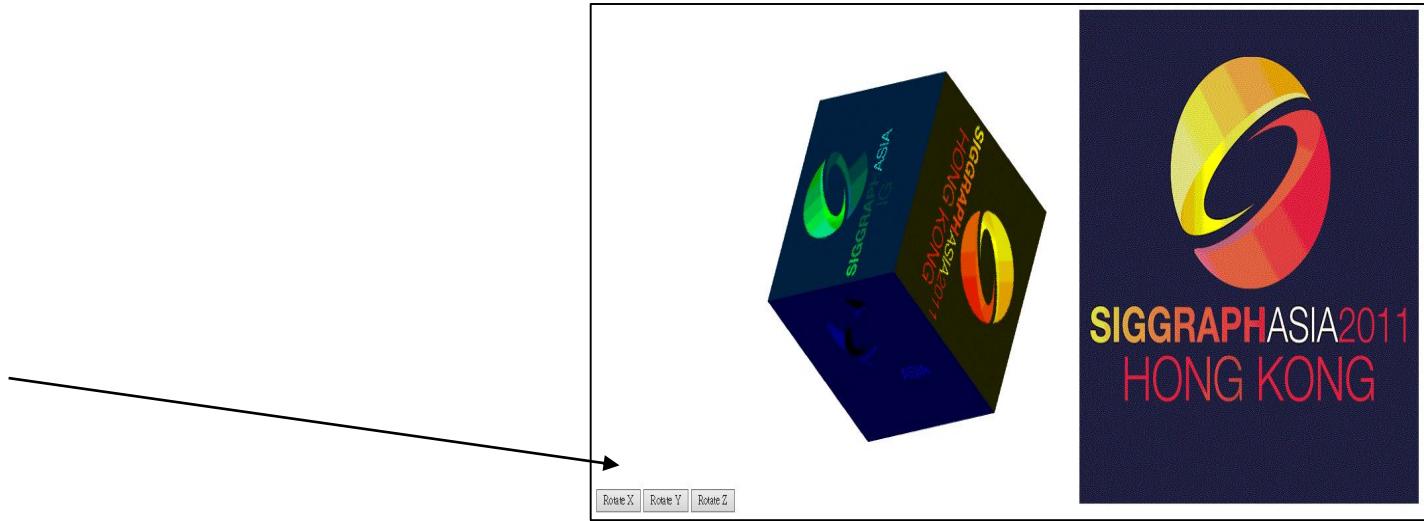
Texture map of a gif image onto cube



textureCube1.html (1/5)

```
<!DOCTYPE html>
<html>
<style type="text/css">
    canvas { background: blue; }
</style>

<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
```



textureCube1.html (2/5)

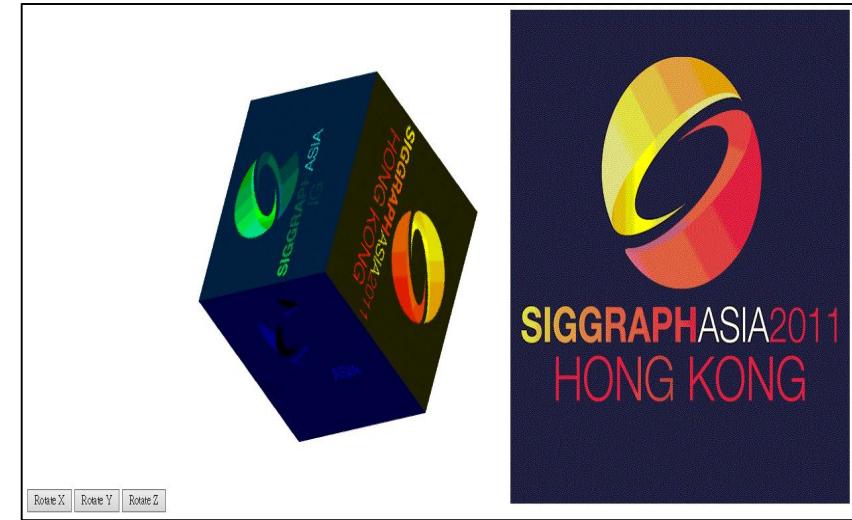
```
<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vColor;
attribute vec2 vTexCoord;

varying vec4 fColor;
varying vec2 fTexCoord;

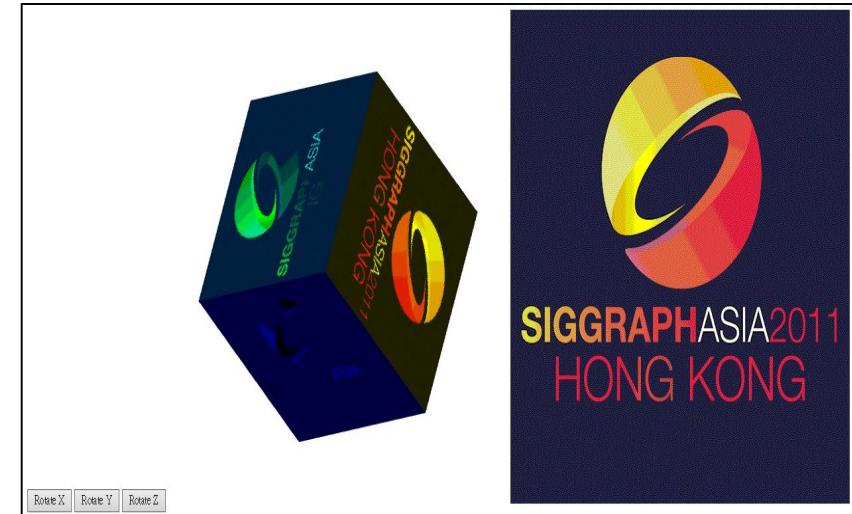
uniform vec3 theta;

void main()
{
    // Compute the sines and cosines of theta for each of
    // the three axes in one computation.
    vec3 angles = radians( theta );
    vec3 c = cos( angles );
    vec3 s = sin( angles );
```



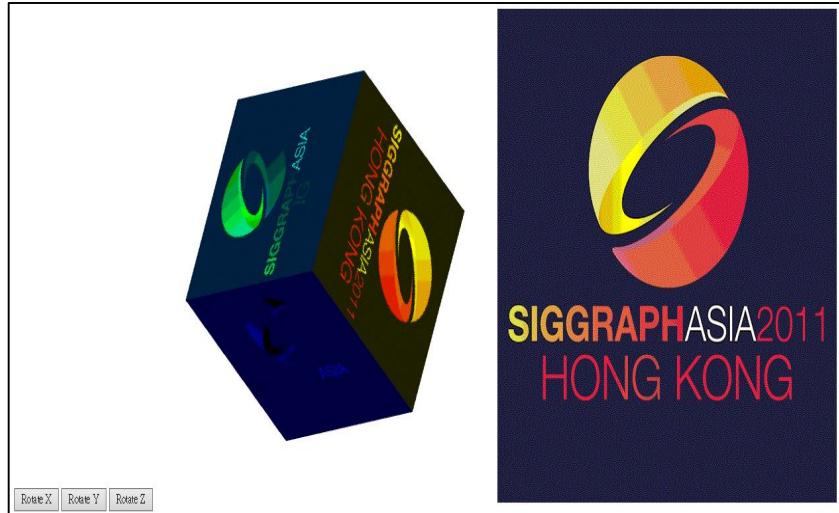
textureCube1.html (3/5)

```
// Remember: these matrices are column-major
mat4 rx = mat4( 1.0, 0.0, 0.0, 0.0,
                  0.0, c.x, s.x, 0.0,
                  0.0, -s.x, c.x, 0.0,
                  0.0, 0.0, 0.0, 1.0 );
mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                  0.0, 1.0, 0.0, 0.0,
                  s.y, 0.0, c.y, 0.0,
                  0.0, 0.0, 0.0, 1.0 );
mat4 rz = mat4( c.z, -s.z, 0.0, 0.0,
                  s.z, c.z, 0.0, 0.0,
                  0.0, 0.0, 1.0, 0.0,
                  0.0, 0.0, 0.0, 1.0 );
fColor = vColor;
fTexCoord = vTexCoord;
gl_Position = rz * ry * rx * vPosition;
}
</script>
```



textureCube1.html (4/5)

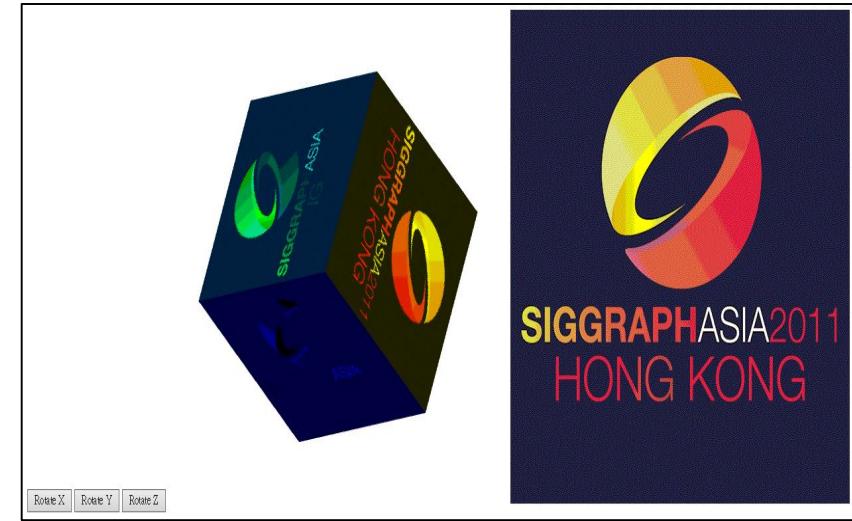
```
<script id="fragment-shader" type="x-shader/x-fragment">  
  
precision mediump float;  
  
varying vec4 fColor;  
varying vec2 fTexCoord;  
  
uniform sampler2D texture;  
  
void  
main()  
{  
    gl_FragColor = fColor * texture2D( texture, fTexCoord );  
}  
</script>
```



textureCube1.html (5/5)

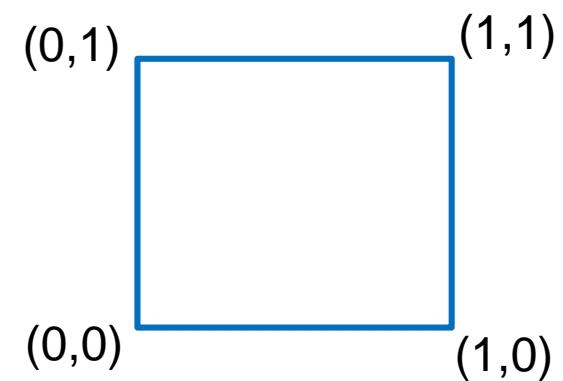
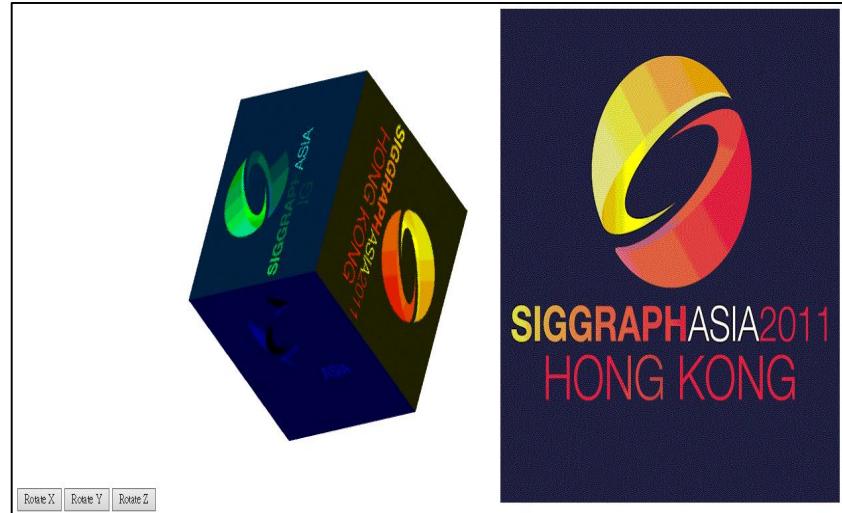
```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="textureCube1.js"></script>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
<img id = "texImage" src = "SA2011_black.gif"></img>
</body>
</html>
```



textureCube1.js (1/11)

```
var canvas;  
var gl;  
  
var numVertices = 36;  
  
var texSize = 64;  
  
var program;  
  
var pointsArray = [];  
var colorsArray = [];  
var texCoordsArray = [];  
  
var texture;  
  
var texCoord = [ vec2(0, 0), vec2(0, 1), vec2(1, 1), vec2(1, 0) ];
```



textureCube1.js (2/11)

```
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 )  
];  
  
var vertexColors = [  
    vec4( 0.0, 0.0, 0.0, 1.0 ), // black  
    vec4( 1.0, 0.0, 0.0, 1.0 ), // red  
    vec4( 1.0, 1.0, 0.0, 1.0 ), // yellow  
    vec4( 0.0, 1.0, 0.0, 1.0 ), // green  
    vec4( 0.0, 0.0, 1.0, 1.0 ), // blue  
    vec4( 1.0, 0.0, 1.0, 1.0 ), // magenta  
    vec4( 0.0, 1.0, 1.0, 1.0 ), // white  
    vec4( 0.0, 1.0, 1.0, 1.0 ) // cyan  
];
```



textureCube1.js (3/11)

```
var xAxis = 0;  
var yAxis = 1;  
var zAxis = 2;  
var axis = xAxis;  
var theta = [45.0, 45.0, 45.0];  
  
var thetaLoc;  
  
function configureTexture( image ) {  
    texture = gl.createTexture();  
    gl.bindTexture( gl.TEXTURE_2D, texture );  
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);  
    gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image );  
    gl.generateMipmap( gl.TEXTURE_2D );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );  
  
    gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);  
}
```

Specify the pixel storage modes.

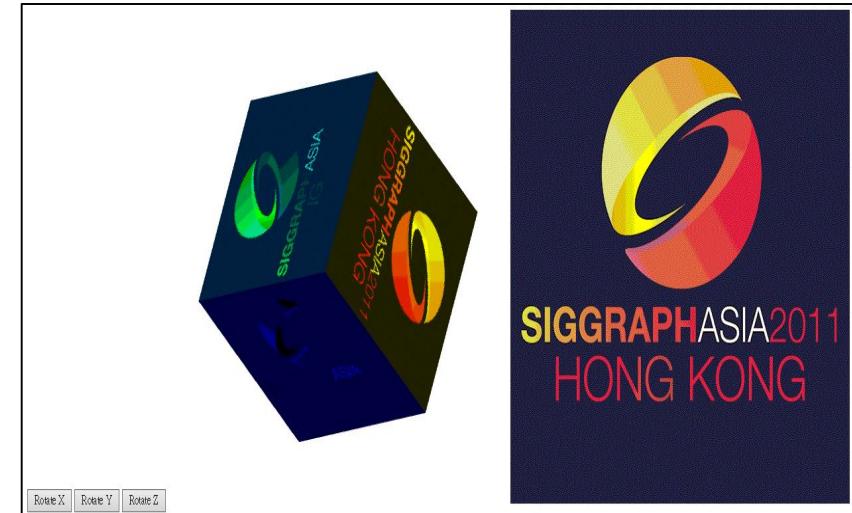
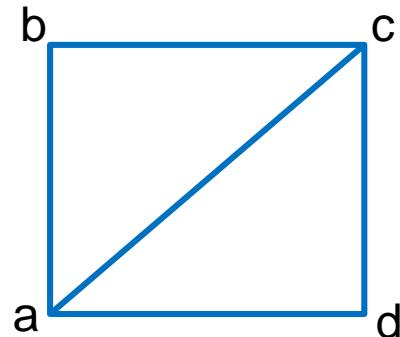
Flip the source data along its vertical axis if true



Rotate X Rotate Y Rotate Z

textureCube1.js (4/11)

```
function quad(a, b, c, d) {  
    pointsArray.push(vertices[a]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[0]);  
  
    pointsArray.push(vertices[b]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[1]);  
  
    pointsArray.push(vertices[c]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[2]);
```



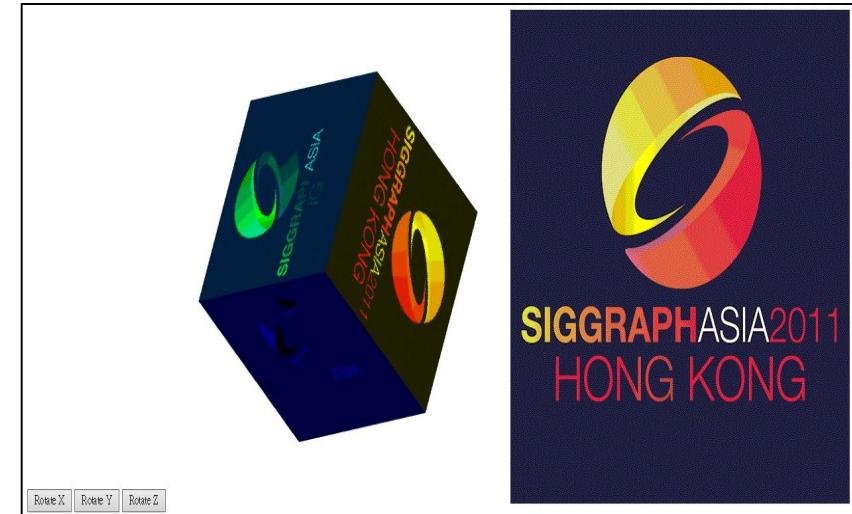
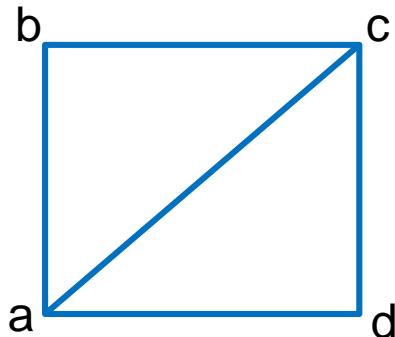
textureCube1.js (5/11)

```
pointsArray.push(vertices[a]);  
colorsArray.push(vertexColors[a]);  
texCoordsArray.push(texCoord[0]);
```

```
pointsArray.push(vertices[c]);  
colorsArray.push(vertexColors[a]);  
texCoordsArray.push(texCoord[2]);
```

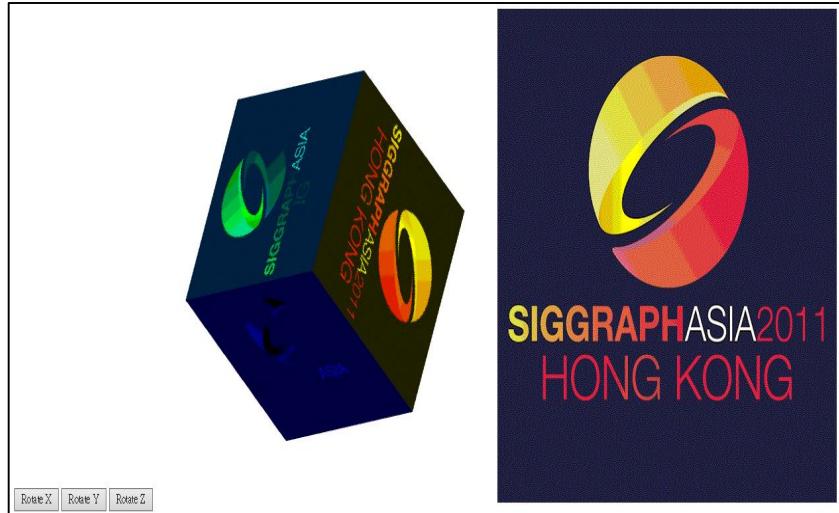
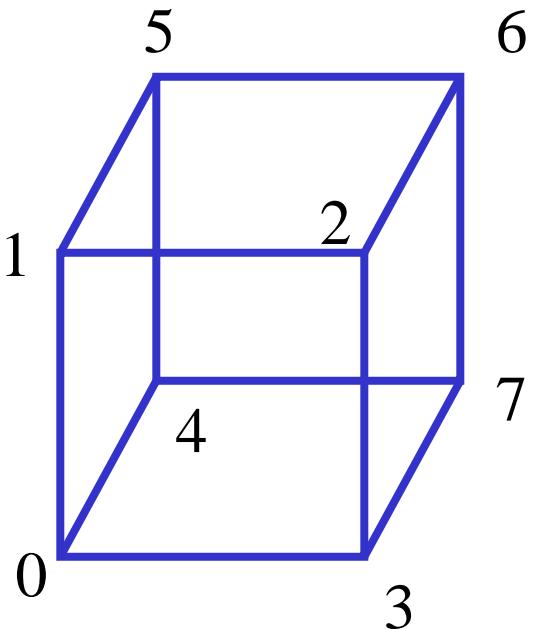
```
pointsArray.push(vertices[d]);  
colorsArray.push(vertexColors[a]);  
texCoordsArray.push(texCoord[3]);
```

```
} // end of quad(a,b,c,d)
```



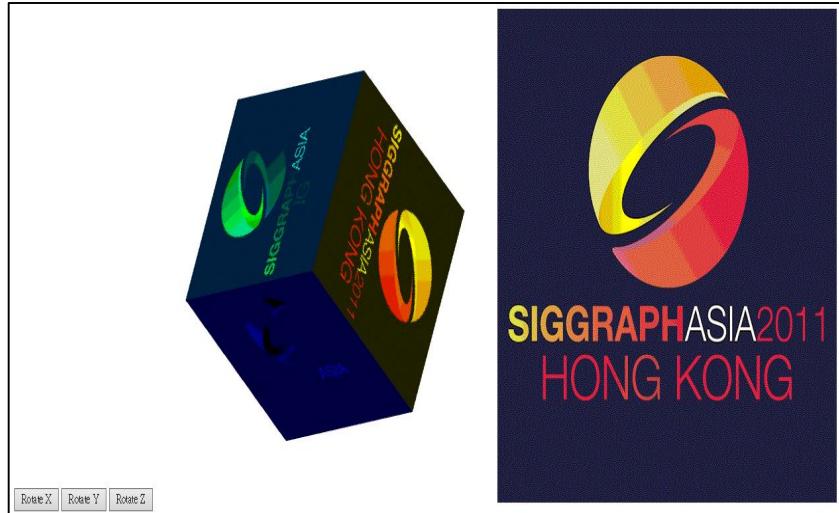
textureCube1.js (6/11)

```
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 );
}
```



textureCube1.js (7/11)

```
window.onload = function init() {  
  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );  
  
    gl.enable(gl.DEPTH_TEST);
```



textureCube1.js (8/11)

```
//  
// Load shaders and initialize attribute buffers  
//  
program = initShaders( gl, "vertex-shader", "fragment-shader" );  
gl.useProgram( program );  
  
colorCube();  
  
var cBuffer = gl.createBuffer();  
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );  
gl.bufferData( gl.ARRAY_BUFFER, flatten(colorsArray), gl.STATIC_DRAW );  
  
var vColor = gl.getAttribLocation( program, "vColor" );  
gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );  
gl.enableVertexAttribArray( vColor );
```

buffer containing vertex attributes, such as vertex coordinates, texture coordinate data, or vertex color data.

An array buffer that will be copied into the data store

contents of the buffer are likely to be used often and not change often

Fixed point values are accessed

the offset in bytes between the beginning of consecutive vertex attributes

an offset in bytes of the first component in the vertex attribute array

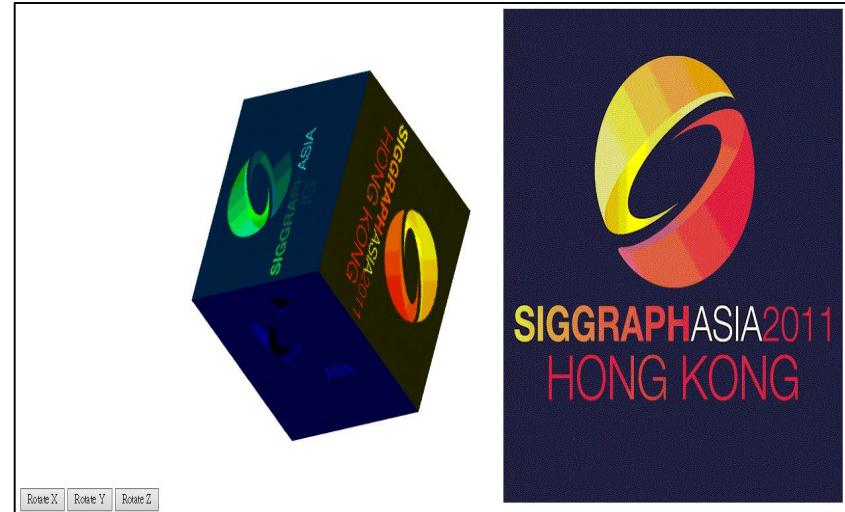
textureCube1.js (9/11)

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

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

var tBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, tBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW );

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



textureCube1.js (10/11)

```
// Initialize a texture

var image = new Image();
image.onload = function() { configureTexture( image ); }
image.src = "SA2011_black.gif"

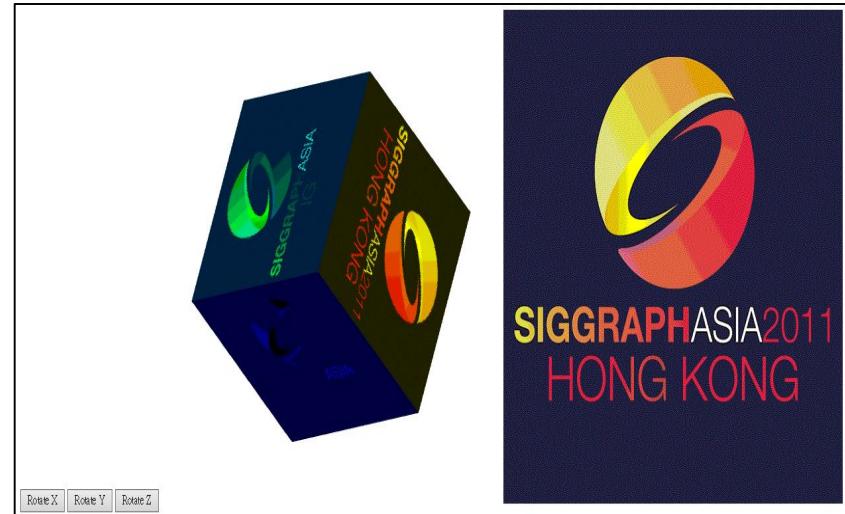
// var image = document.getElementById("texImage");

// configureTexture( image );

thetaLoc = gl.getUniformLocation(program, "theta");

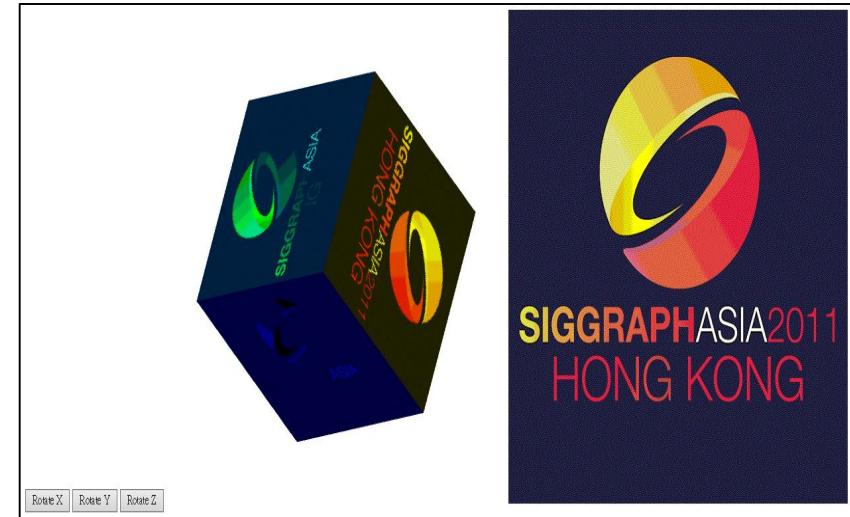
document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};

render();
} // end of window.onload
```



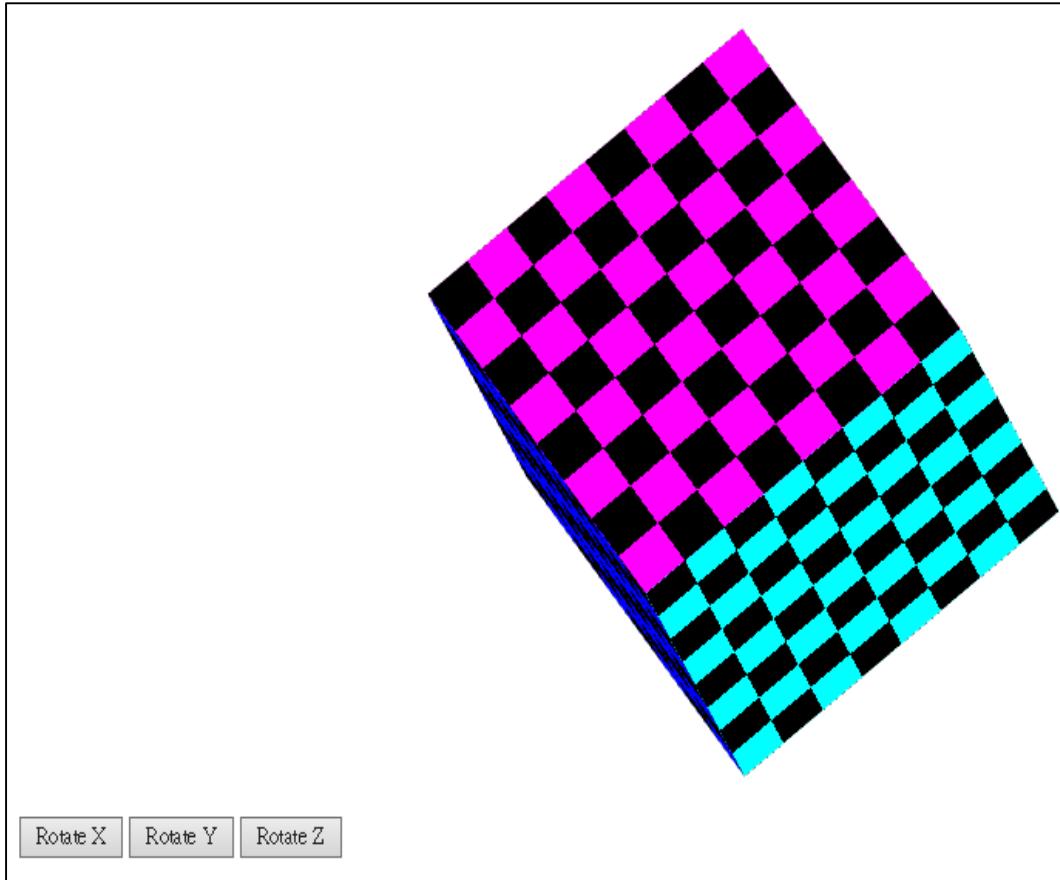
textureCube1.js (11/11)

```
var render = function() {  
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
    theta[axis] += 2.0;  
    gl.uniform3fv(thetaLoc, flatten(theta));  
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );  
    requestAnimFrame(render);  
  
} // end of render()
```



Sample Programs: `textureCubev2.html`, `textureCubev2.js`

Texture mapping checkerboard onto cube



textureCubev2.html (1/5)

```
<!DOCTYPE html>
<html>

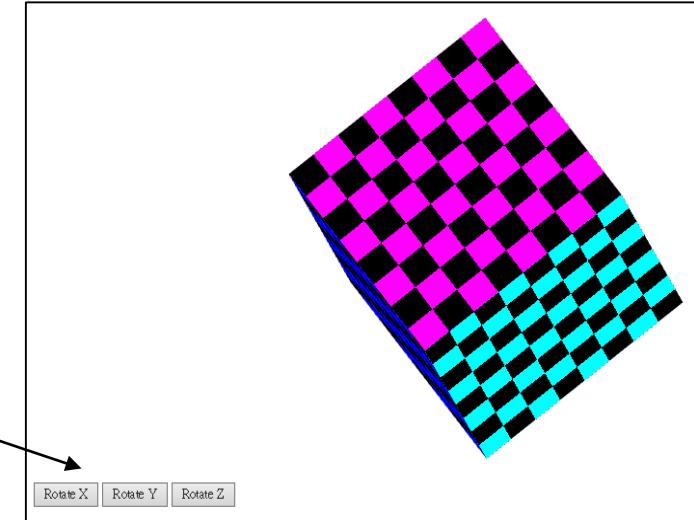
<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vColor;
attribute vec2 vTexCoord;

varying vec4 fColor;
varying vec2 fTexCoord;

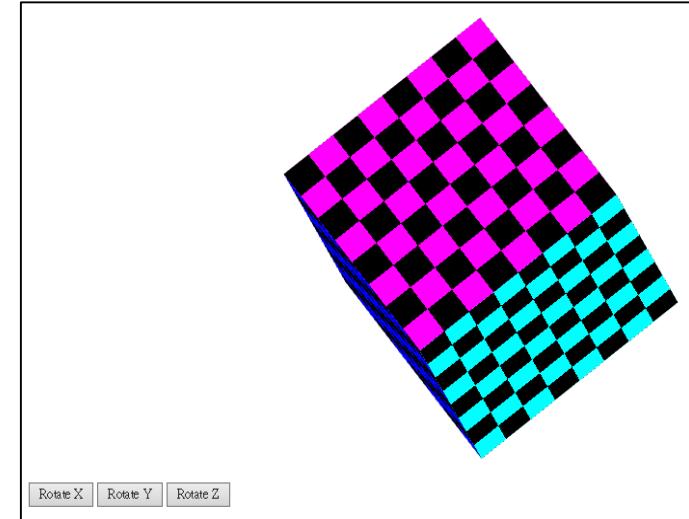
uniform vec3 theta;
```



textureCubev2.html (2/5)

```
void main()
{
    // Compute the sines and cosines of theta for each of
    // the three axes in one computation.
    vec3 angles = radians( theta );
    vec3 c = cos( angles );
    vec3 s = sin( angles );

    // Remember: these matrices are column-major
    mat4 rx = mat4( 1.0, 0.0, 0.0, 0.0,
                    0.0, c.x, s.x, 0.0,
                    0.0, -s.x, c.x, 0.0,
                    0.0, 0.0, 0.0, 1.0 );
    mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                    0.0, 1.0, 0.0, 0.0,
                    s.y, 0.0, c.y, 0.0,
                    0.0, 0.0, 0.0, 1.0 );
```

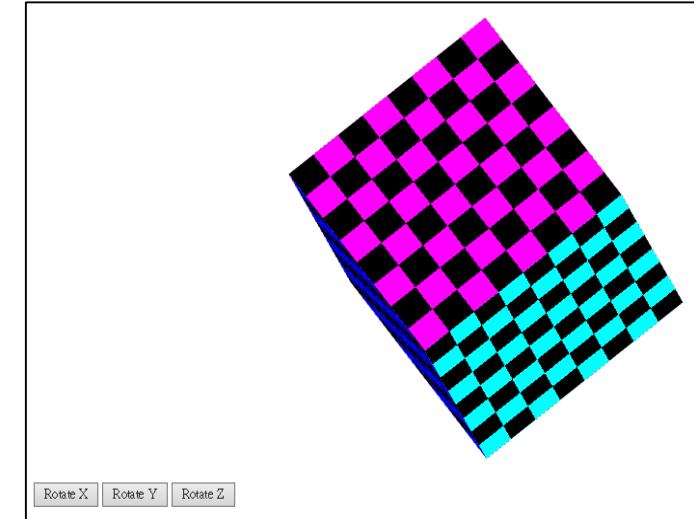


textureCubev2.html (3/5)

```
mat4 rz = mat4( c.z, -s.z, 0.0, 0.0,
                 s.z, c.z, 0.0, 0.0,
                 0.0, 0.0, 1.0, 0.0,
                 0.0, 0.0, 0.0, 1.0 );

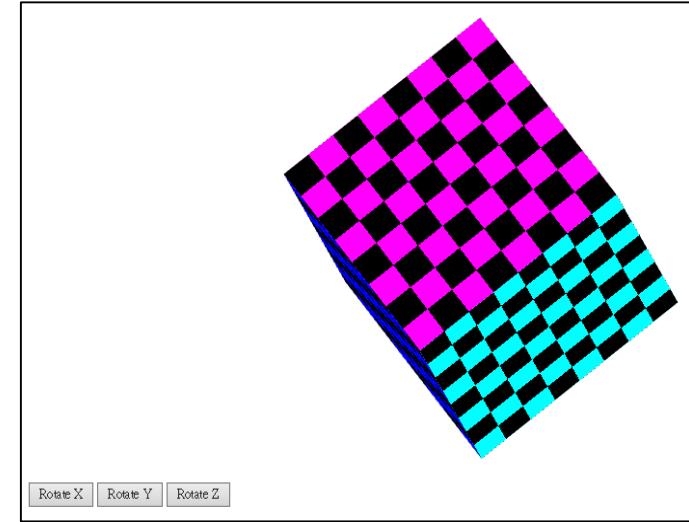
fColor = vColor;
fTexCoord = vTexCoord;
gl_Position = rz * ry * rx * vPosition;
}

</script>
```



textureCubev2.html (4/5)

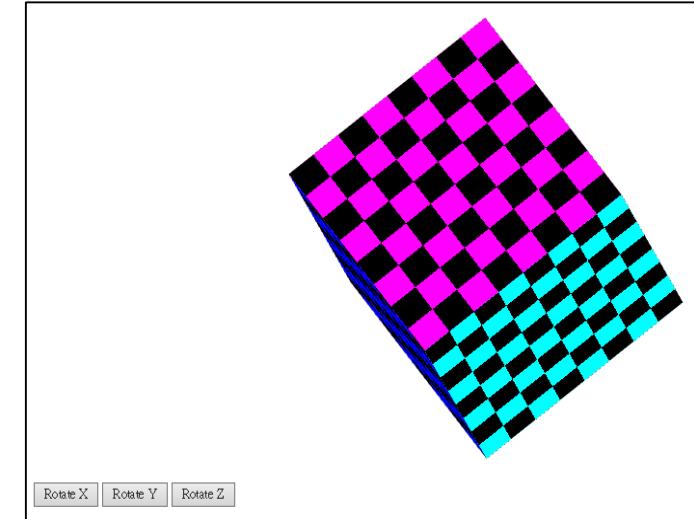
```
<script id="fragment-shader" type="x-shader/x-fragment">  
  
precision mediump float;  
  
varying vec4 fColor;  
varying vec2 fTexCoord;  
  
uniform sampler2D texture;  
  
void  
main()  
{  
    gl_FragColor = fColor*texture2D( texture, fTexCoord );  
}  
</script>
```



textureCubev2.html (5/5)

```
<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="textureCubev2.js"></script>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



textureCubev2.js (1/12)

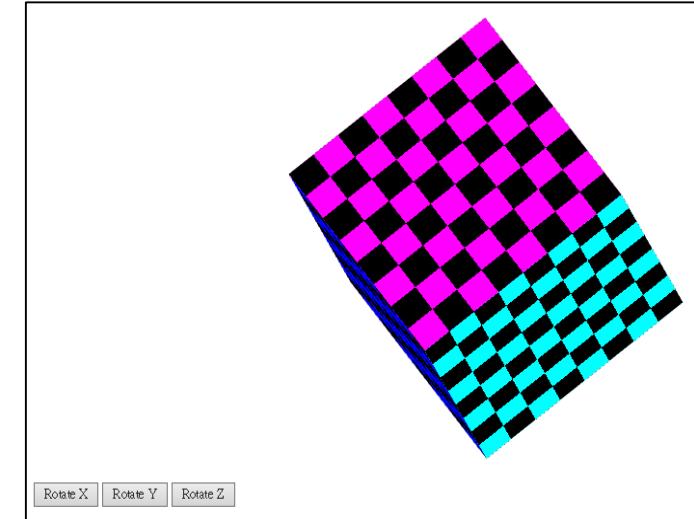
```
var canvas;
var gl;

var numVertices = 36;

var texSize = 64;

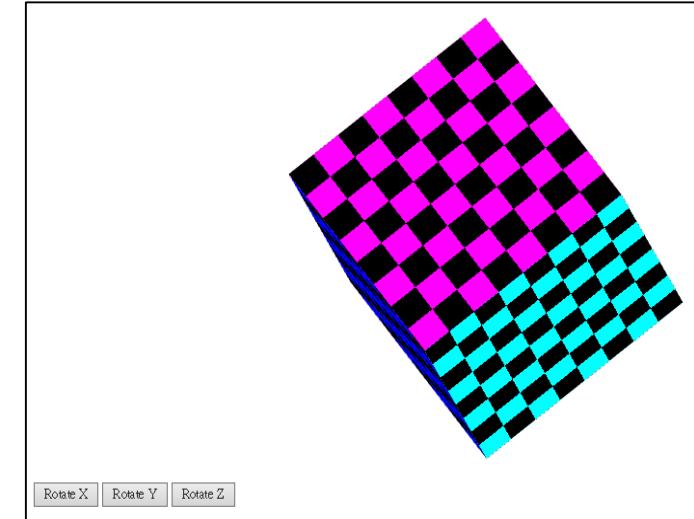
// Create a checkerboard pattern using floats

var image1 = new Array()
for (var i =0; i<texSize; i++) image1[i] = new Array();
for (var i =0; i<texSize; i++)
    for ( var j = 0; j < texSize; j++)
        image1[i][j] = new Float32Array(4);
for (var i =0; i<texSize; i++) for (var j=0; j<texSize; j++) {
    var c = (((i & 0x8) == 0) ^ ((j & 0x8) == 0));
    image1[i][j] = [c, c, c, 1];
}
```



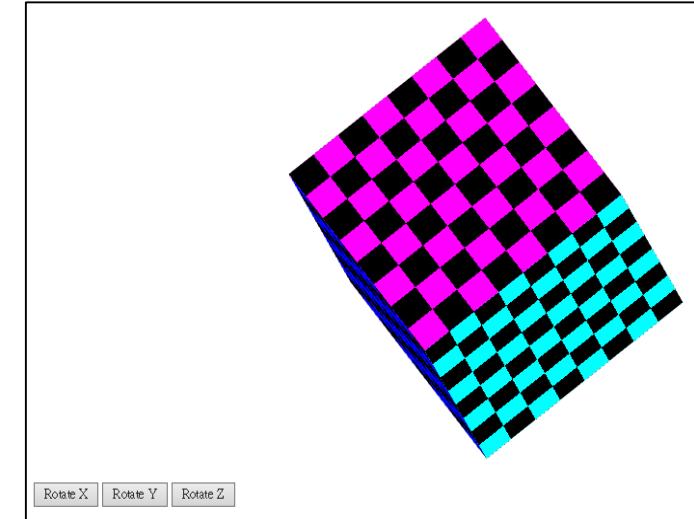
textureCubev2.js (2/12)

```
// Convert floats to ubytes for texture  
  
var image2 = new Uint8Array(4*texSize*texSize);  
  
for ( var i = 0; i < texSize; i++ )  
    for ( var j = 0; j < texSize; j++ )  
        for(var k =0; k<4; k++)  
            image2[4*texSize*i+4*j+k] = 255*image1[i][j][k];  
  
var pointsArray = [];  
var colorsArray = [];  
var texCoordsArray = [];
```



textureCubev2.js (3/12)

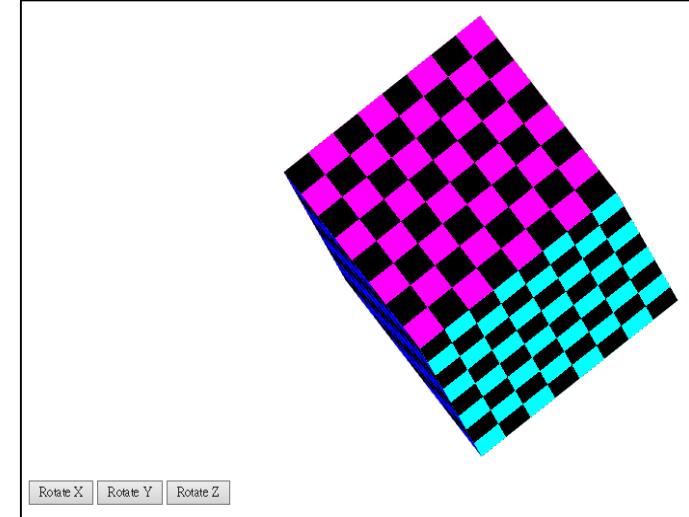
```
var texCoord = [  
    vec2(0, 0),  
    vec2(0, 1),  
    vec2(1, 1),  
    vec2(1, 0)  
];  
  
var vertices = [  
    vec4( -0.5, -0.5,  0.5, 1.0 ),  
    vec4( -0.5,  0.5,  0.5, 1.0 ),  
    vec4(  0.5,  0.5,  0.5, 1.0 ),  
    vec4(  0.5, -0.5,  0.5, 1.0 ),  
    vec4( -0.5, -0.5, -0.5, 1.0 ),  
    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 )  
];
```



textureCubev2.js (4/12)

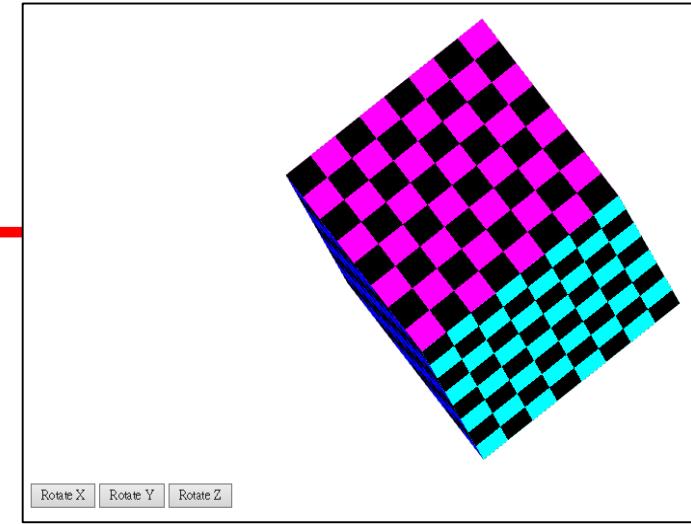
```
var vertexColors = [  
    vec4( 0.0, 0.0, 0.0, 1.0 ), // black  
    vec4( 1.0, 0.0, 0.0, 1.0 ), // red  
    vec4( 1.0, 1.0, 0.0, 1.0 ), // yellow  
    vec4( 0.0, 1.0, 0.0, 1.0 ), // green  
    vec4( 0.0, 0.0, 1.0, 1.0 ), // blue  
    vec4( 1.0, 0.0, 1.0, 1.0 ), // magenta  
    vec4( 0.0, 1.0, 1.0, 1.0 ), // white  
    vec4( 0.0, 1.0, 1.0, 1.0 ) // cyan  
];  
window.onload = init;
```

```
var xAxis = 0;  
var yAxis = 1;  
var zAxis = 2;  
var axis = xAxis;  
var theta = [45.0, 45.0, 45.0];  
var thetaLoc;
```



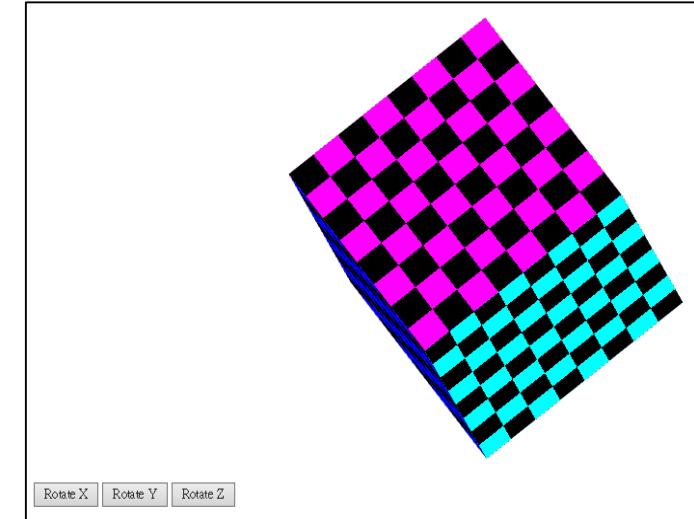
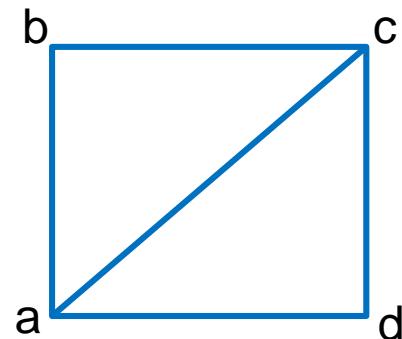
textureCubev2.js (5/12)

```
function configureTexture(image) {  
    texture = gl.createTexture();  
    gl.activeTexture( gl.TEXTURE0 );  
    gl.bindTexture( gl.TEXTURE_2D, texture );  
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);  
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texSize, texSize, 0, gl.RGBA, gl.UNSIGNED_BYTE, image);  
    gl.generateMipmap( gl.TEXTURE_2D );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );  
}
```



textureCubev2.js (6/12)

```
function quad(a, b, c, d) {  
  
    pointsArray.push(vertices[a]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[0]);  
  
    pointsArray.push(vertices[b]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[1]);  
  
    pointsArray.push(vertices[c]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[2]);
```



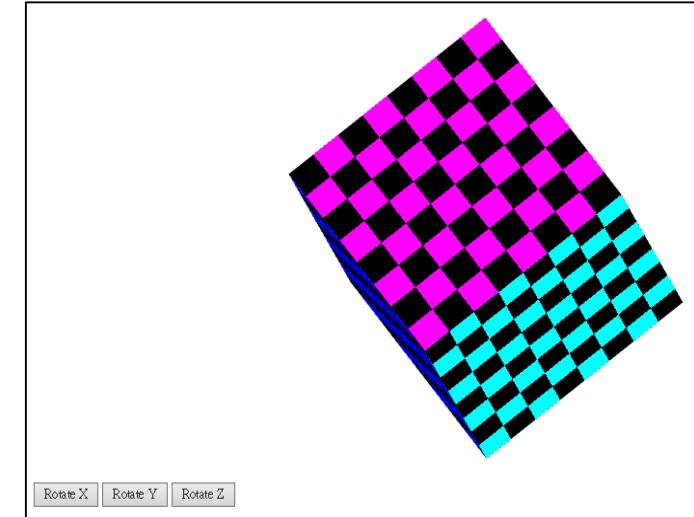
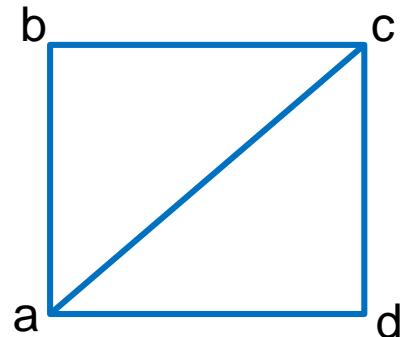
textureCubev2.js (7/12)

```
pointsArray.push(vertices[a]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[0]);

pointsArray.push(vertices[c]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[2]);

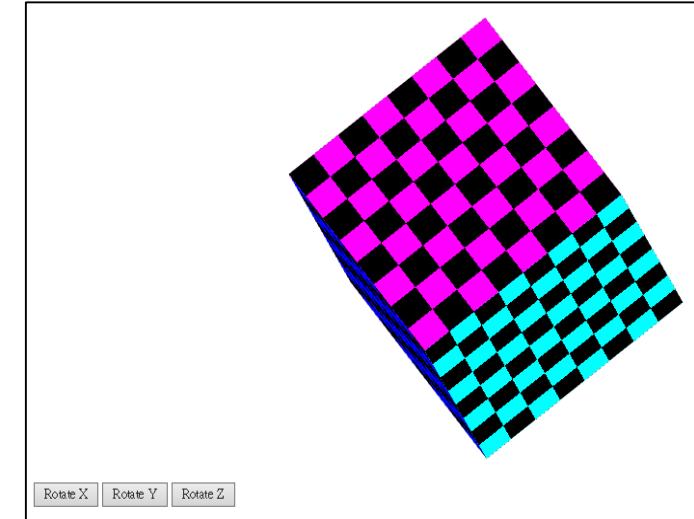
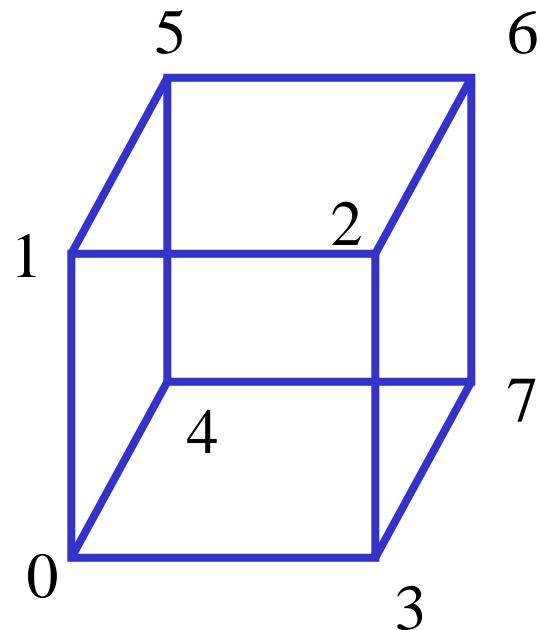
pointsArray.push(vertices[d]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[3]);

} // end of quad(a,b,c,d)
```



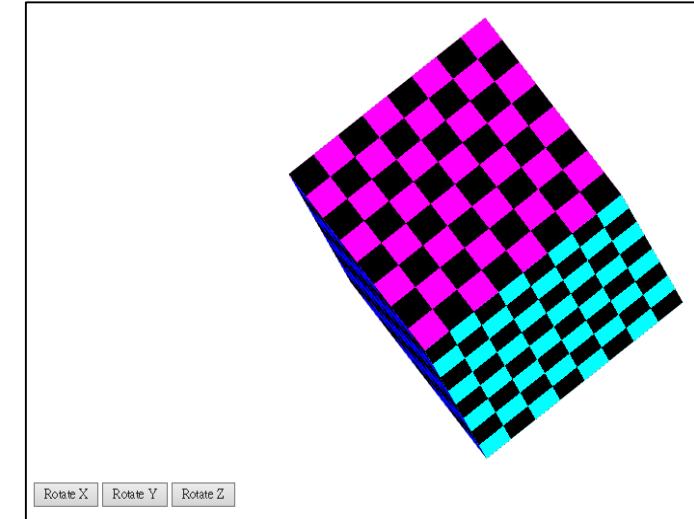
textureCubev2.js (8/12)

```
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 );
}
```



textureCubev2.js (9/12)

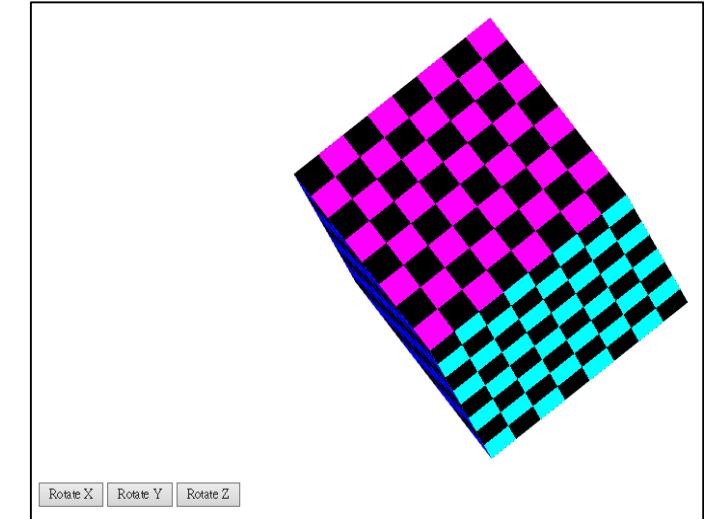
```
function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );  
  
    gl.enable(gl.DEPTH_TEST);  
  
    //  
    // Load shaders and initialize attribute buffers  
    //  
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );  
    gl.useProgram( program );  
  
    colorCube();
```



textureCubev2.js (10/12)

```
var cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(colorsArray), 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(pointsArray), gl.STATIC_DRAW);
var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```



textureCubev2.js (11/12)

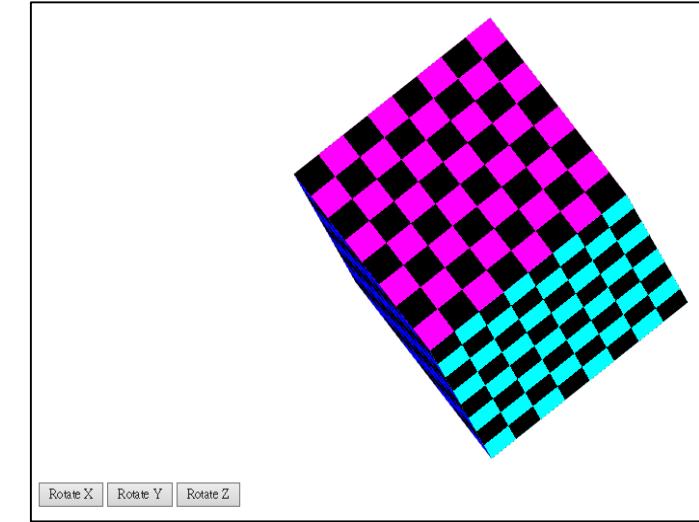
```
var tBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, tBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW );
var vTexCoord = gl.getAttribLocation( program, "vTexCoord");
gl.vertexAttribPointer(vTexCoord, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vTexCoord);

configureTexture(image2);

thetaLoc = gl.getUniformLocation(program, "theta");

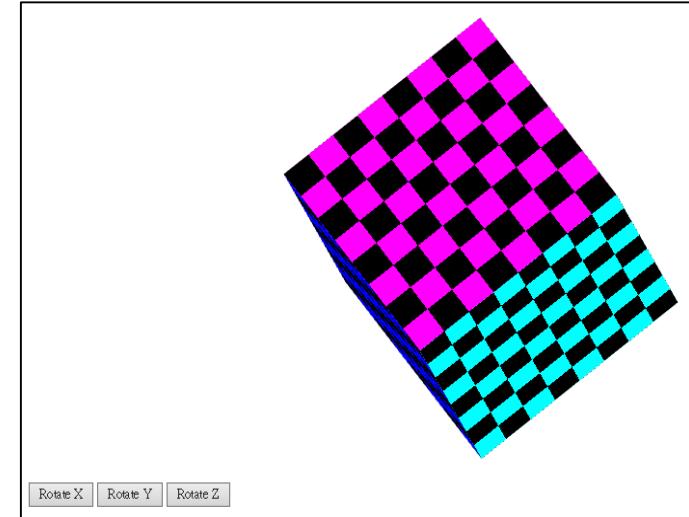
document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};

render();
}
```

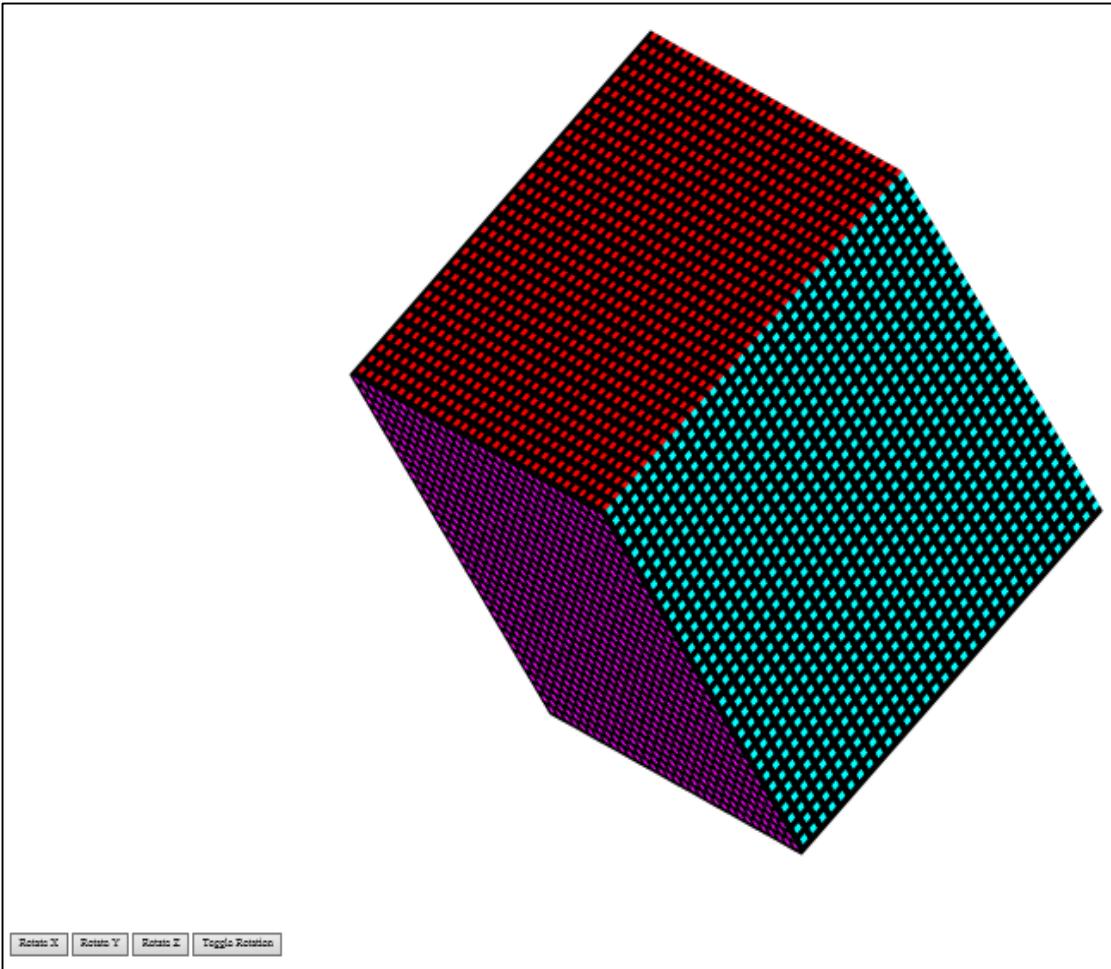


textureCubev2.js (12/12)

```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    theta[axis] += 2.0;
    gl.uniform3fv(thetaLoc, theta);
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );
    requestAnimFrame(render);
} // end of render()
```



Sample Programs: `textureCubev3.html`, `textureCubev3.js`



Texture map onto cube using
two texture images multiplied
together in fragment shader

textureCubev3.html (1/5)

```
<!DOCTYPE html>
<html>

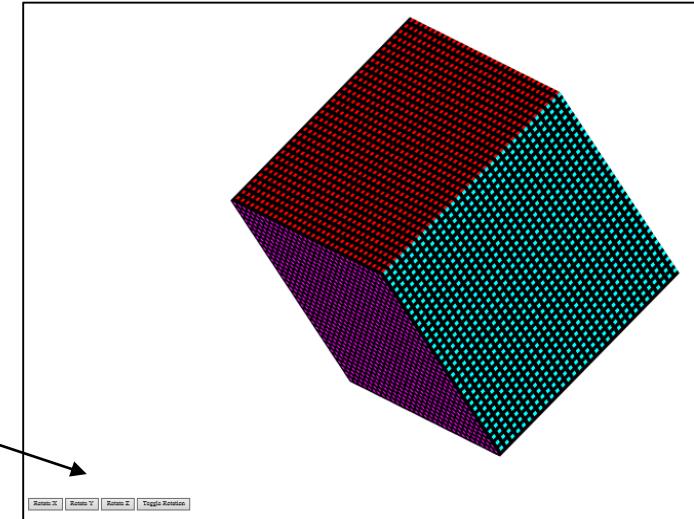
<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
<button id = "ButtonT">Toggle Rotation</button>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vColor;
attribute vec2 vTexCoord;

varying vec4 fColor;
varying vec2 fTexCoord;

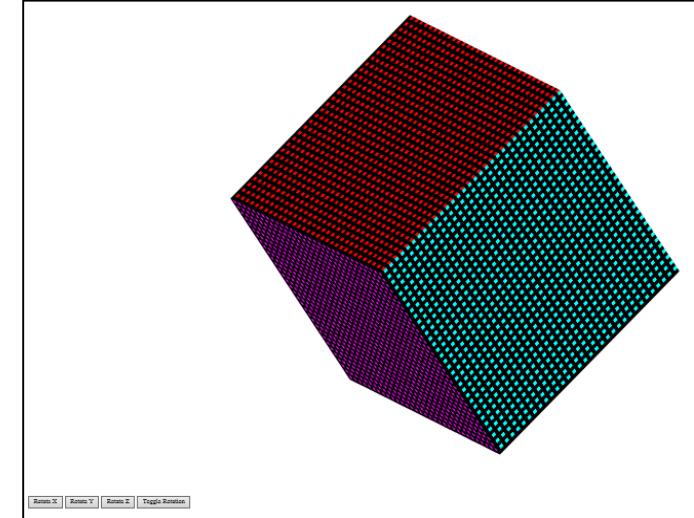
uniform vec3 theta;
```



textureCubev3.html (2/5)

```
void main()
{
    // Compute the sines and cosines of theta for each of
    // the three axes in one computation.
    vec3 angles = radians( theta );
    vec3 c = cos( angles );
    vec3 s = sin( angles );

    // Remember: these matrices are column-major
    mat4 rx = mat4( 1.0, 0.0, 0.0, 0.0,
                    0.0, c.x, s.x, 0.0,
                    0.0, -s.x, c.x, 0.0,
                    0.0, 0.0, 0.0, 1.0 );
    mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                    0.0, 1.0, 0.0, 0.0,
                    s.y, 0.0, c.y, 0.0,
                    0.0, 0.0, 0.0, 1.0 );
```

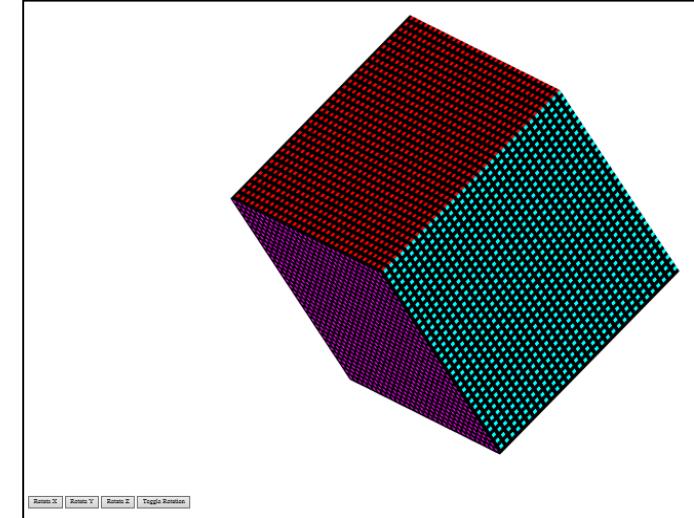


textureCubev3.html (3/5)

```
mat4 rz = mat4( c.z, -s.z, 0.0, 0.0,
                 s.z, c.z, 0.0, 0.0,
                 0.0, 0.0, 1.0, 0.0,
                 0.0, 0.0, 0.0, 1.0 );

fColor = vColor;
fTexCoord = vTexCoord;
gl_Position = rz * ry * rx * vPosition;
}

</script>
```



textureCubev3.html (4/5)

```
<script id="fragment-shader" type="x-shader/x-fragment">

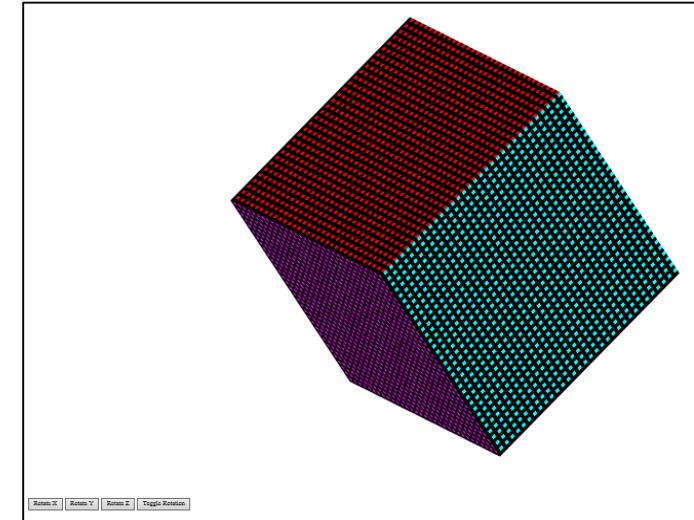
precision mediump float;

varying vec4 fColor;
varying vec2 fTexCoord;

uniform sampler2D Tex0;
uniform sampler2D Tex1;

void
main()
{
    gl_FragColor = fColor*(texture2D(Tex0, fTexCoord)*texture2D(Tex1, fTexCoord));
}

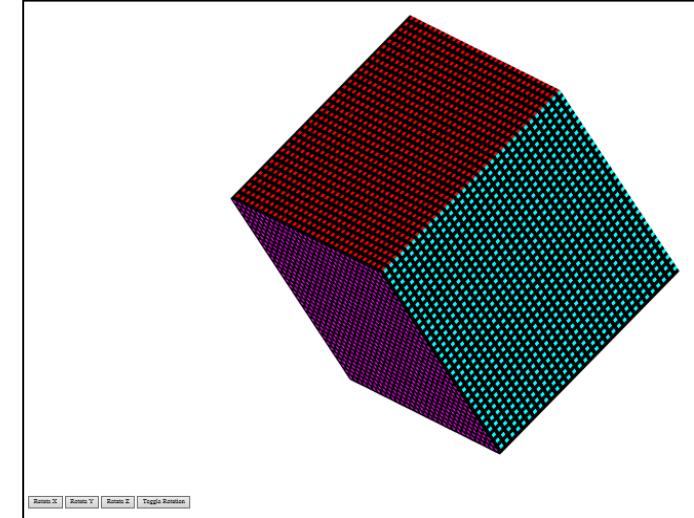
</script>
```



textureCubev3.html (5/5)

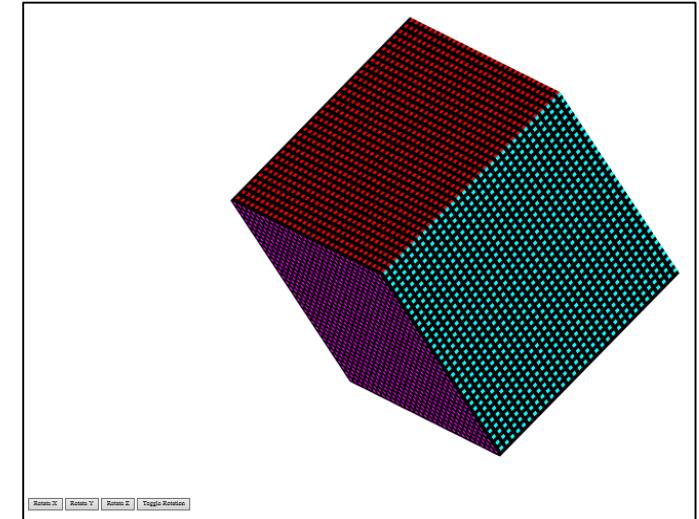
```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="textureCubev3.js"></script>

<body>
<canvas id="gl-canvas" width="1024" height="1024">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



textureCubev3.js (1/15)

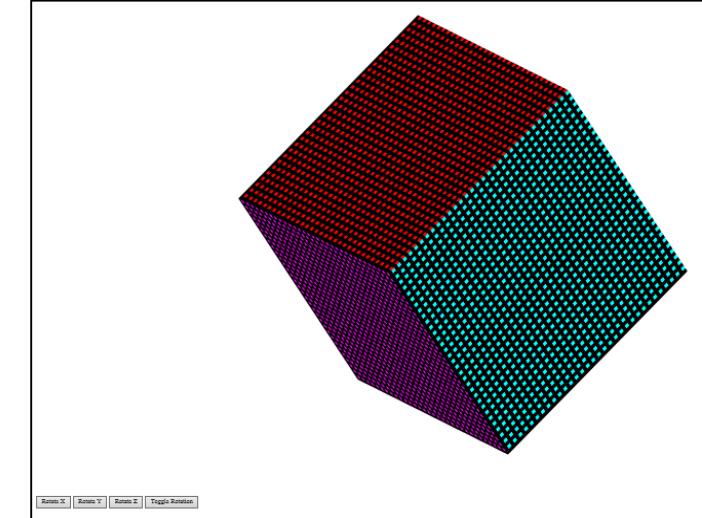
```
var canvas;  
var gl;  
  
var numVertices = 36;  
  
var texSize = 256;  
var numChecks = 64;  
  
var program;  
  
var texture1, texture2;  
var t1, t2;  
  
var c;  
  
var flag = true;
```



textureCubev3.js (2/15)

```
var image1 = new Uint8Array(4*texSize*texSize);

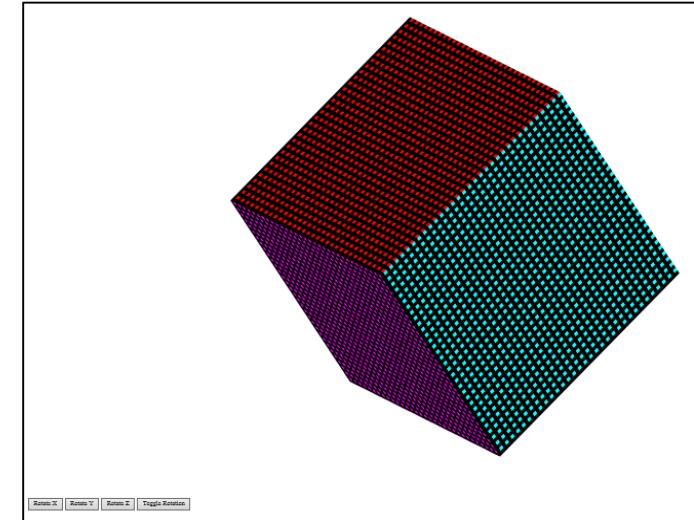
for ( var i = 0; i < texSize; i++ ) {
    for ( var j = 0; j <texSize; j++ ) {
        var patchx = Math.floor( i / (texSize/numChecks) );
        if(patchx%2) c = 255;
        else c = 0;
        image1[4*i*texSize+4*j] = c;
        image1[4*i*texSize+4*j+1] = c;
        image1[4*i*texSize+4*j+2] = c;
        image1[4*i*texSize+4*j+3] = 255;
    }
}
```



textureCubev3.js (3/15)

```
var image2 = new Uint8Array(4*texSize*texSize);

// Create a checkerboard pattern
for ( var i = 0; i < texSize; i++ ) {
    for ( var j = 0; j < texSize; j++ ) {
        var patchy = Math.floor( j / (texSize/numChecks) );
        if(patchy%2) c = 255;
        else c = 0;
        image2[4*i*texSize+4*j] = c;
        image2[4*i*texSize+4*j+1] = c;
        image2[4*i*texSize+4*j+2] = c;
        image2[4*i*texSize+4*j+3] = 255;
    }
}
```

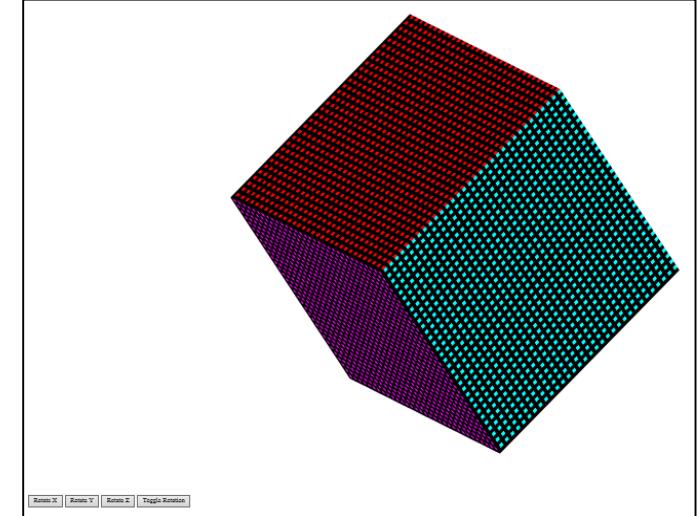


textureCubev3.js (4/15)

```
var pointsArray = [];
var colorsArray = [];
var texCoordsArray = [];

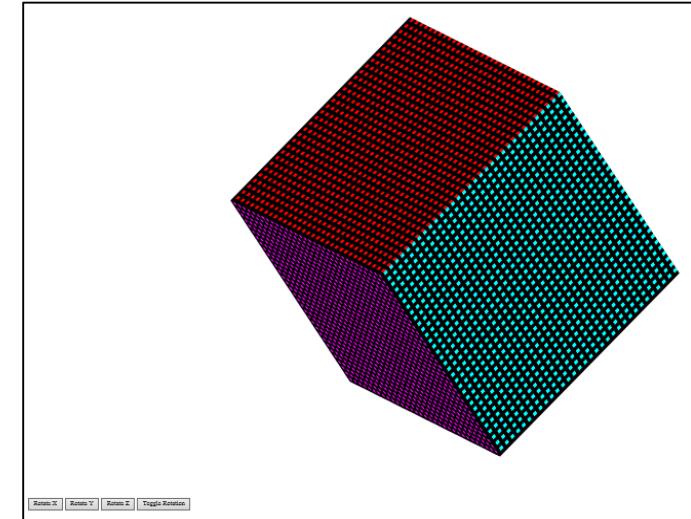
var texCoord = [
    vec2(0, 0),
    vec2(0, 1),
    vec2(1, 1),
    vec2(1, 0)
];

```



textureCubev3.js (5/15)

```
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 )  
];
```

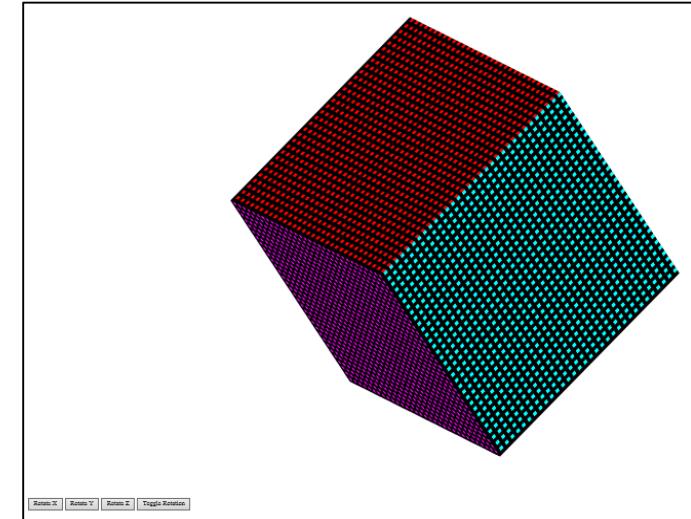


textureCubev3.js (6/15)

```
var vertexColors = [  
    vec4( 0.0, 0.0, 0.0, 1.0 ), // black  
    vec4( 1.0, 0.0, 0.0, 1.0 ), // red  
    vec4( 1.0, 1.0, 0.0, 1.0 ), // yellow  
    vec4( 0.0, 1.0, 0.0, 1.0 ), // green  
    vec4( 0.0, 0.0, 1.0, 1.0 ), // blue  
    vec4( 1.0, 0.0, 1.0, 1.0 ), // magenta  
    vec4( 0.0, 1.0, 1.0, 1.0 ), // white  
    vec4( 0.0, 1.0, 1.0, 1.0 ) // cyan  
];
```

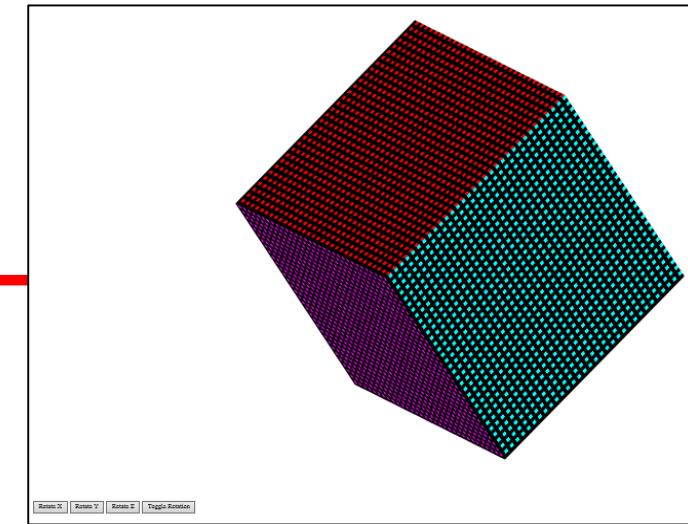
```
var xAxis = 0;  
var yAxis = 1;  
var zAxis = 2;  
var axis = xAxis;
```

```
var theta = [45.0, 45.0, 45.0];  
var thetaLoc;
```



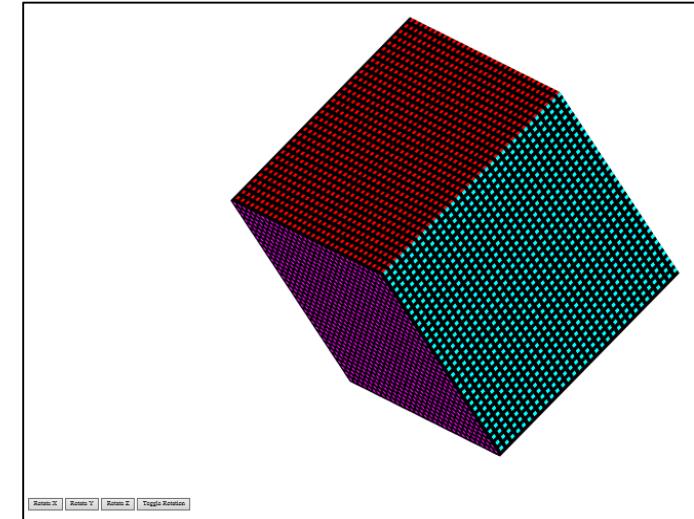
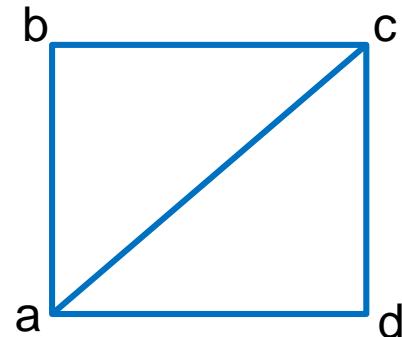
textureCubev3.js (7/15)

```
function configureTexture() {  
    texture1 = gl.createTexture();  
    gl.bindTexture( gl.TEXTURE_2D, texture1 );  
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);  
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texSize, texSize, 0, gl.RGBA, gl.UNSIGNED_BYTE, image1);  
    gl.generateMipmap( gl.TEXTURE_2D );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);  
  
    texture2 = gl.createTexture();  
    gl.bindTexture( gl.TEXTURE_2D, texture2 );  
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);  
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texSize, texSize, 0, gl.RGBA, gl.UNSIGNED_BYTE, image2);  
    gl.generateMipmap( gl.TEXTURE_2D );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);  
} // end of configureTexture()
```



textureCubev3.js (8/15)

```
function quad(a, b, c, d) {  
    pointsArray.push(vertices[a]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[0]);  
  
    pointsArray.push(vertices[b]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[1]);  
  
    pointsArray.push(vertices[c]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[2]);
```



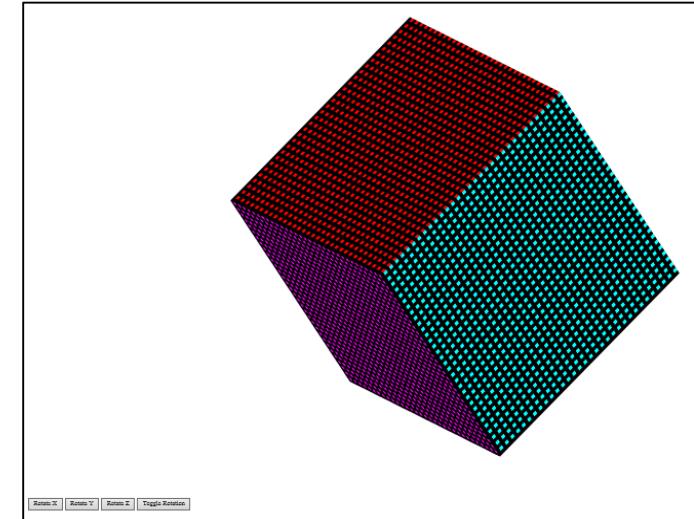
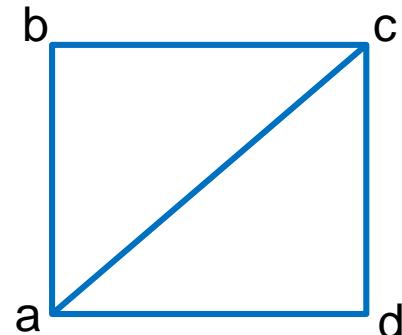
textureCubev3.js (9/15)

```
pointsArray.push(vertices[a]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[0]);

pointsArray.push(vertices[c]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[2]);

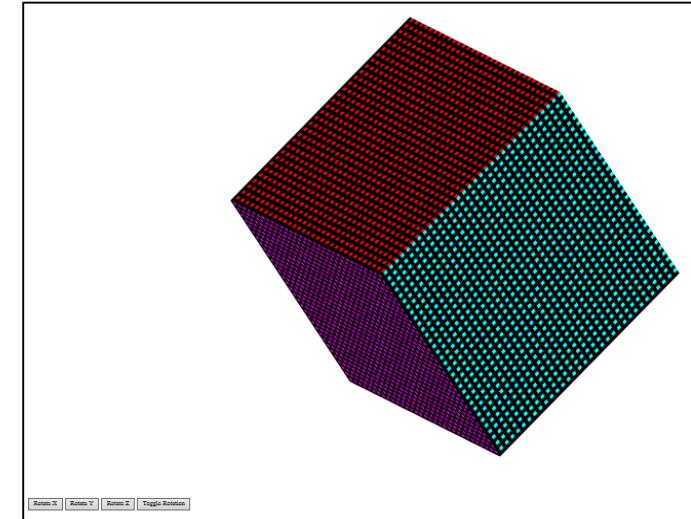
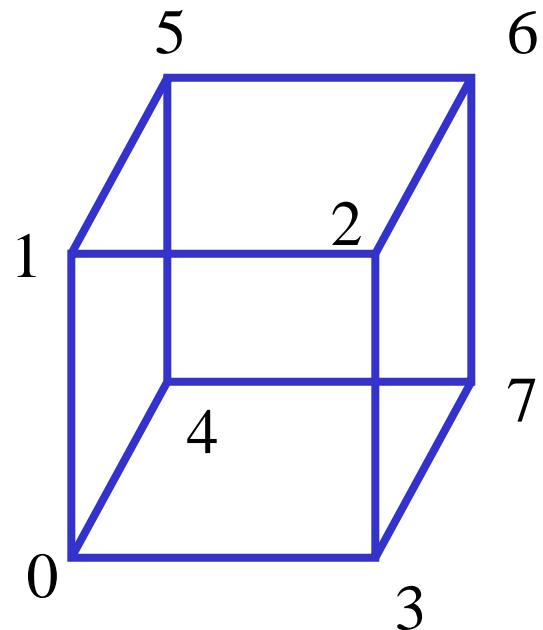
pointsArray.push(vertices[d]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[3]);

} // end of quad(a,b,c,d)
```



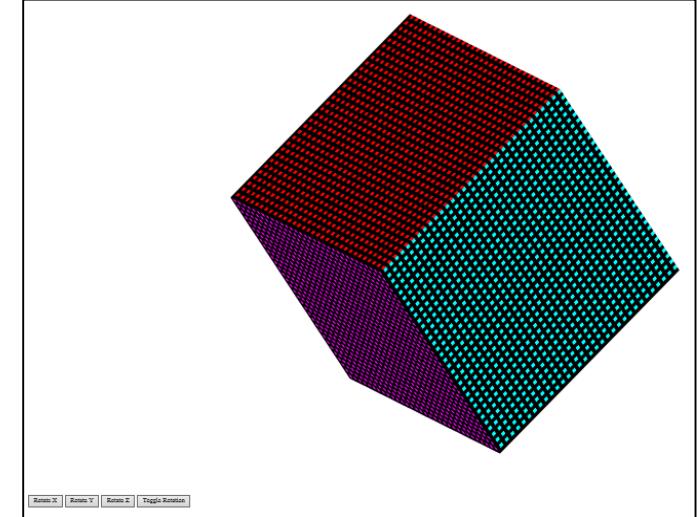
textureCubev3.js (10/15)

```
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 );
}
```



textureCubev3.js (11/15)

```
window.onload = function init() {  
  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );  
  
    gl.enable(gl.DEPTH_TEST);  
  
    // Load shaders and initialize attribute buffers  
  
    program = initShaders( gl, "vertex-shader", "fragment-shader" );  
    gl.useProgram( program );  
  
    colorCube();
```



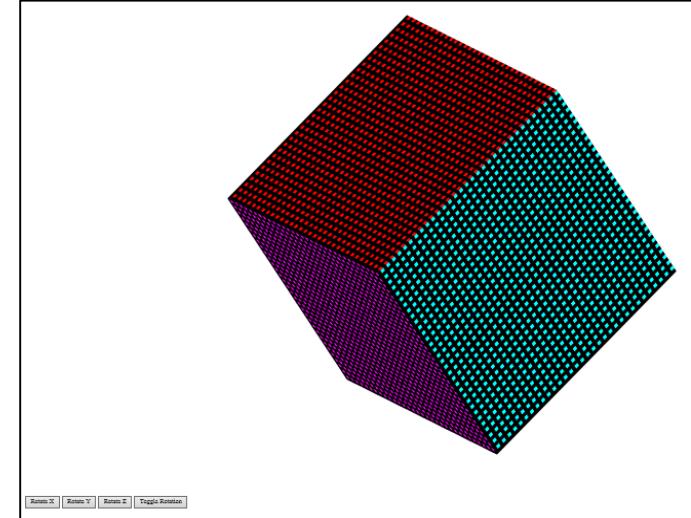
textureCubev3.js (12/15)

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

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



textureCubev3.js (13/15)

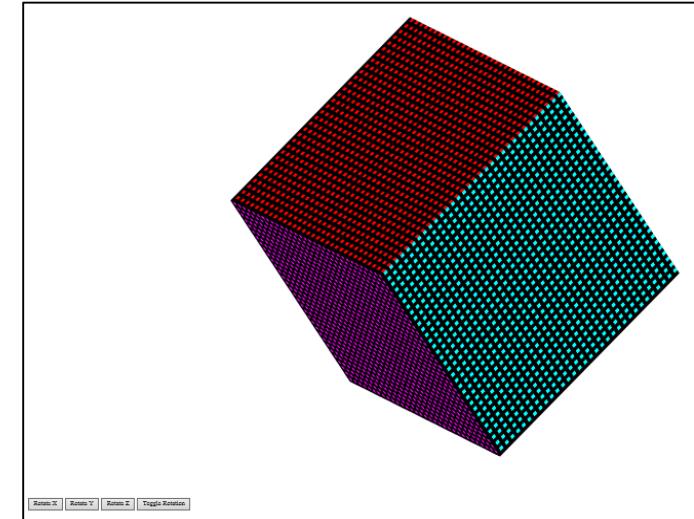
```
var tBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, tBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW );

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

configureTexture();
gl.activeTexture( gl.TEXTURE0 );
gl.bindTexture( gl.TEXTURE_2D, texture1 );
gl.uniform1i(gl.getUniformLocation( program, "Tex0"), 0);

gl.activeTexture( gl.TEXTURE1 );
gl.bindTexture( gl.TEXTURE_2D, texture2 );
gl.uniform1i(gl.getUniformLocation( program, "Tex1"), 1);

thetaLoc = gl.getUniformLocation(program, "theta");
```

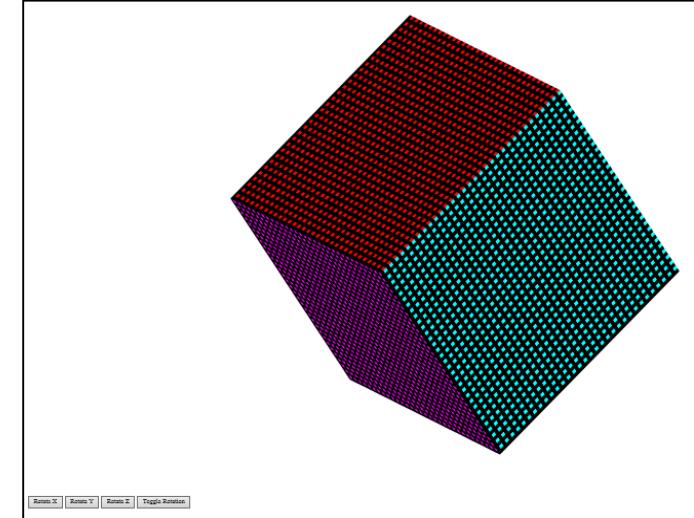


textureCubev3.js (14/15)

```
document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};
document.getElementById("ButtonT").onclick = function() {flag = !flag;};

render();

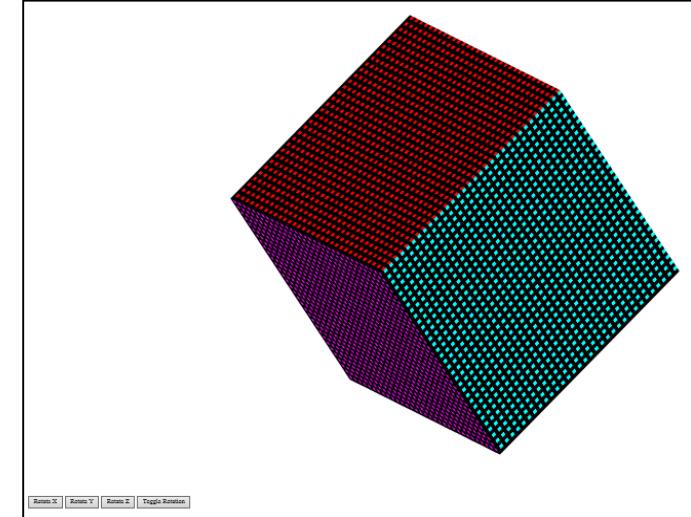
} // end of window.onload
```



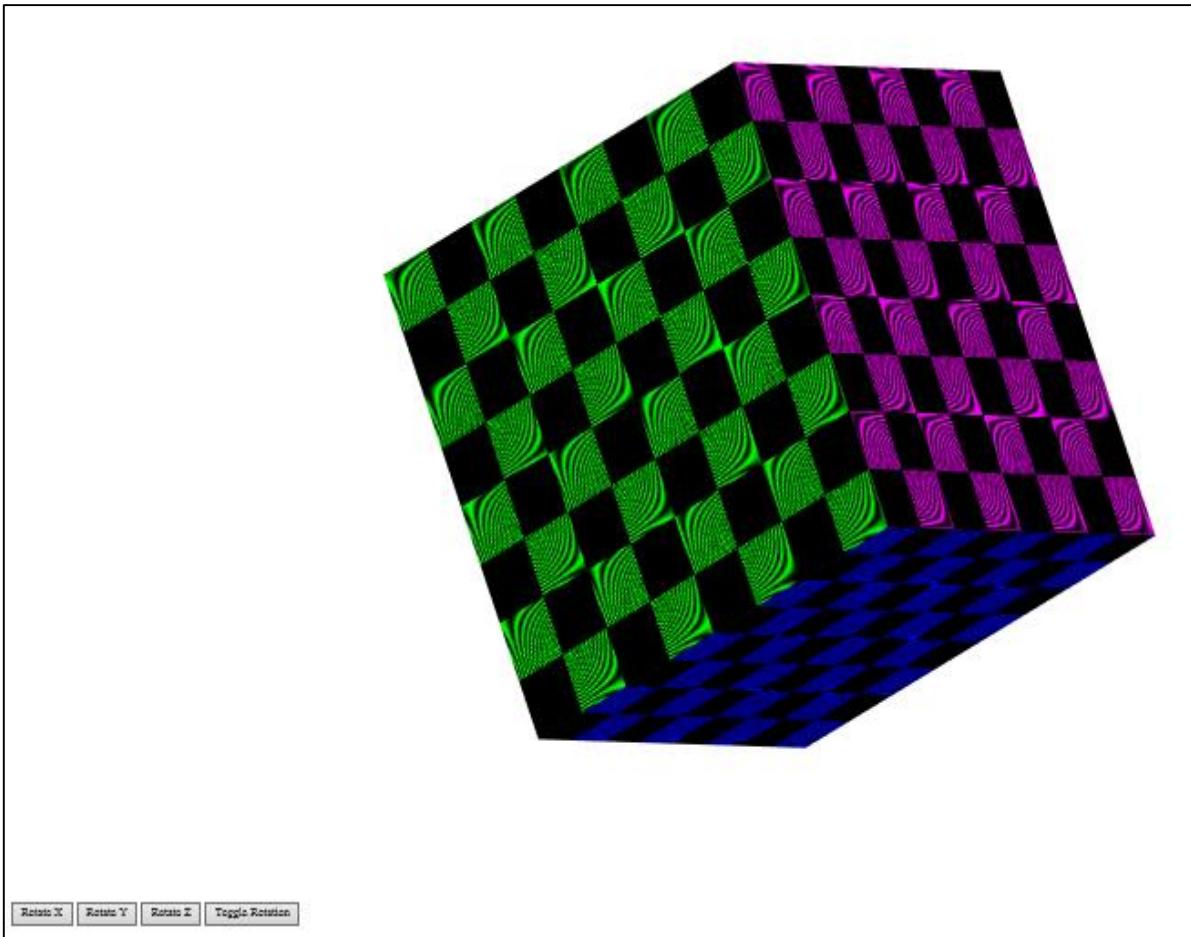
textureCubev3.js (15/15)

```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    if(flag) theta[axis] += 2.0;
    gl.uniform3fv(thetaLoc, theta);
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );
    requestAnimFrame(render);
}

} // end of render()
```



Sample Programs: `textureCubev4.html`, `textureCubev4.js`



Texture map with two texture units. First applies checkerboard, second a sinusoid.

textureCubev4.html (1/5)

```
<!DOCTYPE html>
<html>

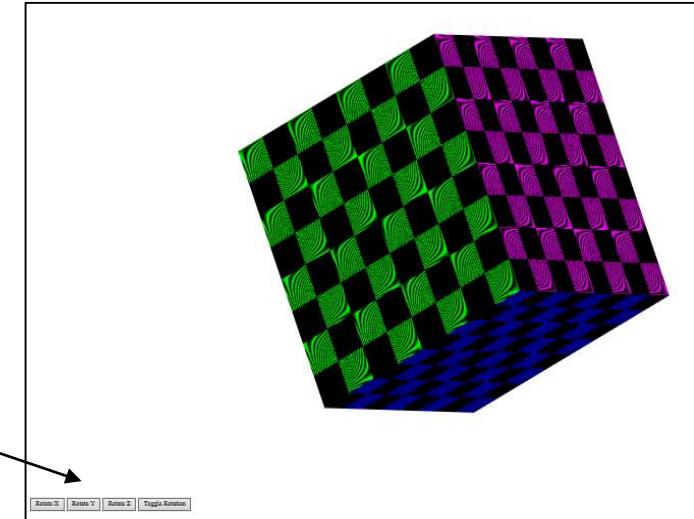
<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
<button id = "ButtonT">Toggle Rotation</button>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vColor;
attribute vec2 vTexCoord;

varying vec4 fColor;
varying vec2 fTexCoord;

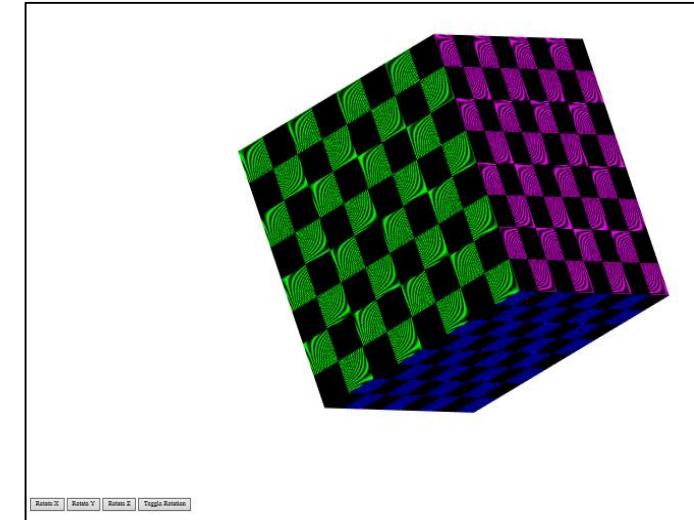
uniform vec3 theta;
```



textureCubev4.html (2/5)

```
void main()
{
    // Compute the sines and cosines of theta for each of
    // the three axes in one computation.
    vec3 angles = radians( theta );
    vec3 c = cos( angles );
    vec3 s = sin( angles );

    // Remember: these matrices are column-major
    mat4 rx = mat4( 1.0, 0.0, 0.0, 0.0,
                    0.0, c.x, s.x, 0.0,
                    0.0, -s.x, c.x, 0.0,
                    0.0, 0.0, 0.0, 1.0 );
    mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                    0.0, 1.0, 0.0, 0.0,
                    s.y, 0.0, c.y, 0.0,
                    0.0, 0.0, 0.0, 1.0 );
```

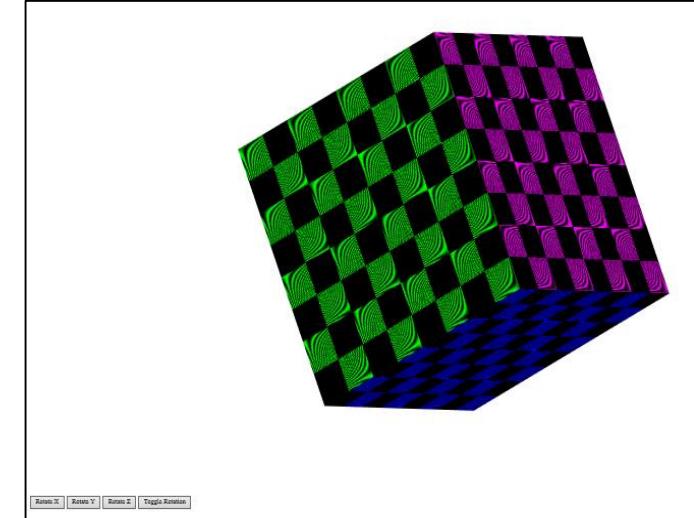


textureCubev4.html (3/5)

```
mat4 rz = mat4( c.z, -s.z, 0.0, 0.0,
                 s.z, c.z, 0.0, 0.0,
                 0.0, 0.0, 1.0, 0.0,
                 0.0, 0.0, 0.0, 1.0 );

fColor = vColor;
fTexCoord = vTexCoord;
gl_Position = rz * ry * rx * vPosition;
}

</script>
```



textureCubev4.html (4/5)

```
<script id="fragment-shader" type="x-shader/x-fragment">

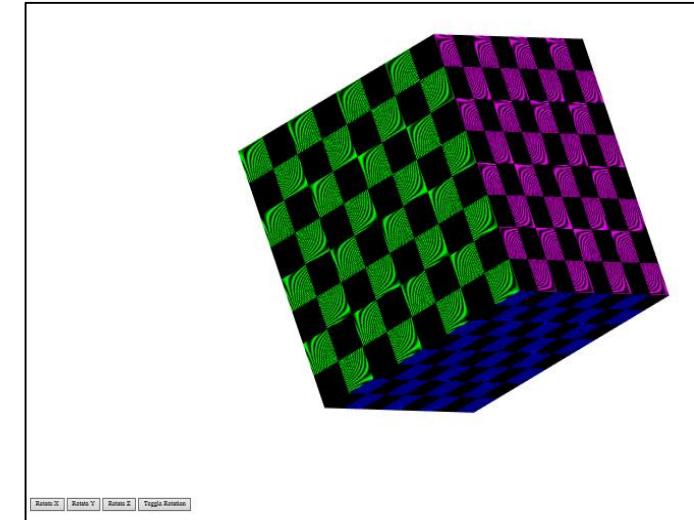
precision mediump float;

varying vec4 fColor;
varying vec2 fTexCoord;

uniform sampler2D Tex0;
uniform sampler2D Tex1;

void
main()
{
    gl_FragColor = fColor * (texture2D(Tex0, fTexCoord) * texture2D(Tex1, fTexCoord));
}

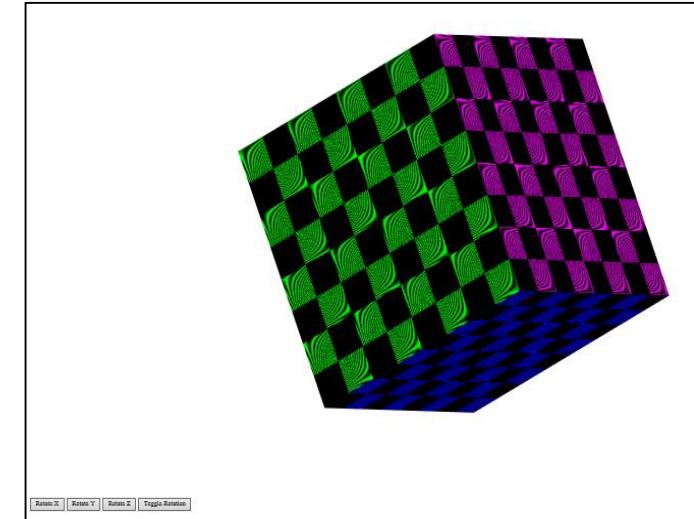
</script>
```



textureCubev4.html (5/5)

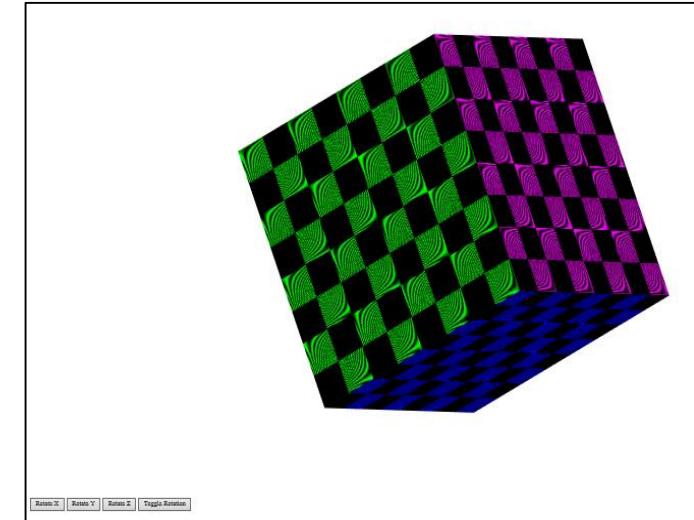
```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="textureCubev4.js"></script>

<body>
<canvas id="gl-canvas" width="1024" height="1024">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



textureCubev4.js (1/15)

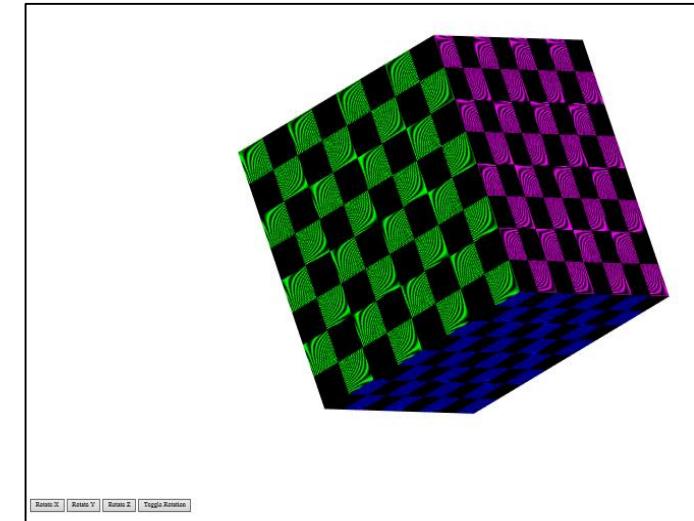
```
var canvas;  
var gl;  
  
var numVertices = 36;  
  
var texSize = 256;  
var numChecks = 8;  
  
var program;  
  
var texture1, texture2;  
var t1, t2;  
  
var c;  
  
var flag = true;
```



textureCubev4.js (2/15)

```
var image1 = new Uint8Array(4*texSize*texSize);

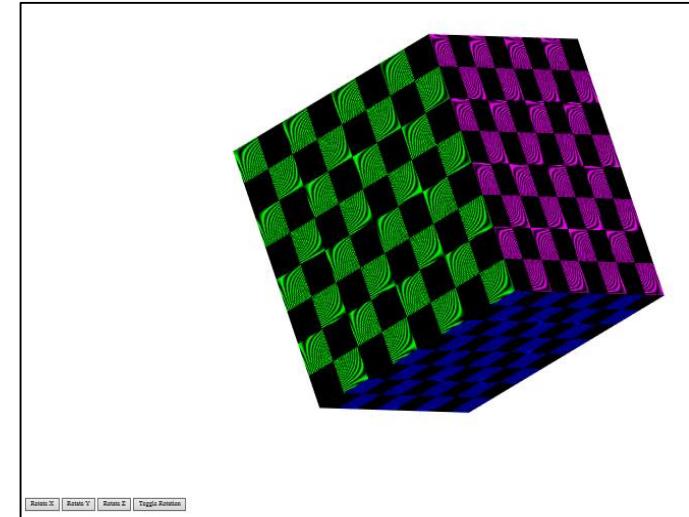
for ( var i = 0; i < texSize; i++ ) {
    for ( var j = 0; j <texSize; j++ ) {
        var patchx = Math.floor( i / (texSize/numChecks) );
        var patchy = Math.floor( j / (texSize/numChecks) );
        if(patchx%2 ^ patchy%2) c = 255;
            else c = 0;
        //c = 255*((i & 0x8) == 0) ^ ((j & 0x8) == 0))
        image1[4*i*texSize+4*j] = c;
        image1[4*i*texSize+4*j+1] = c;
        image1[4*i*texSize+4*j+2] = c;
        image1[4*i*texSize+4*j+3] = 255;
    }
}
```



textureCubev4.js (3/15)

```
var image2 = new Uint8Array(4*texSize*texSize);

// Create a checkerboard pattern
for ( var i = 0; i < texSize; i++ ) {
    for ( var j = 0; j <texSize; j++ ) {
        image2[4*i*texSize+4*j]    = 127+127*Math.sin(0.1*i*j);
        image2[4*i*texSize+4*j+1] = 127+127*Math.sin(0.1*i*j);
        image2[4*i*texSize+4*j+2] = 127+127*Math.sin(0.1*i*j);
        image2[4*i*texSize+4*j+3] = 255;
    }
}
```

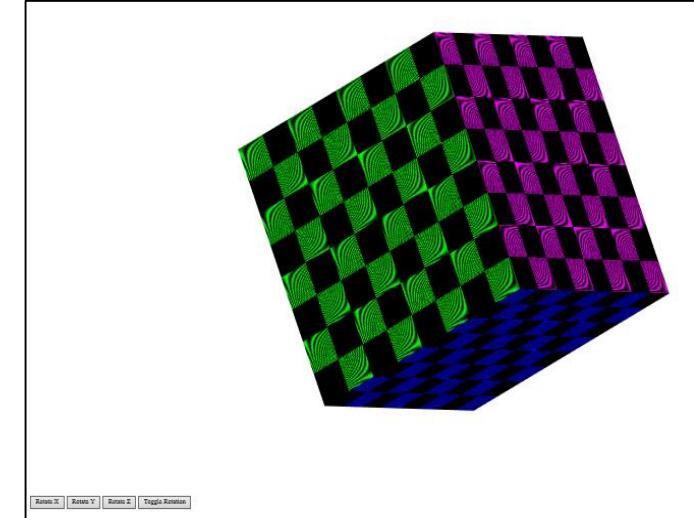


textureCubev4.js (4/15)

```
var pointsArray = [];
var colorsArray = [];
var texCoordsArray = [];

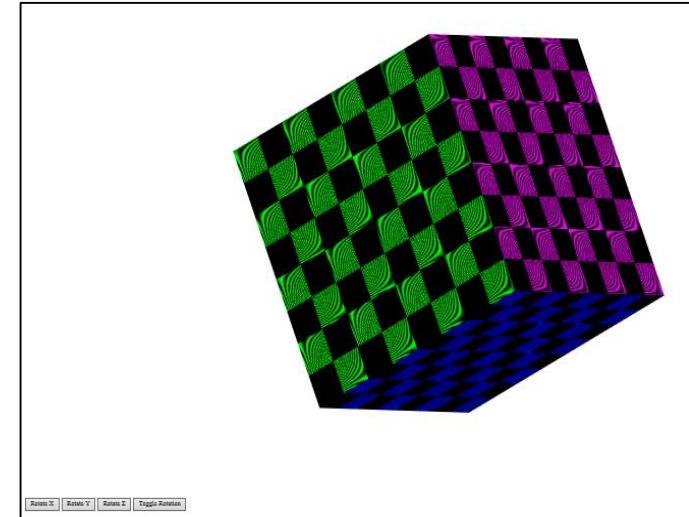
var texCoord = [
    vec2(0, 0),
    vec2(0, 1),
    vec2(1, 1),
    vec2(1, 0)
];

```



textureCubev4.js (5/15)

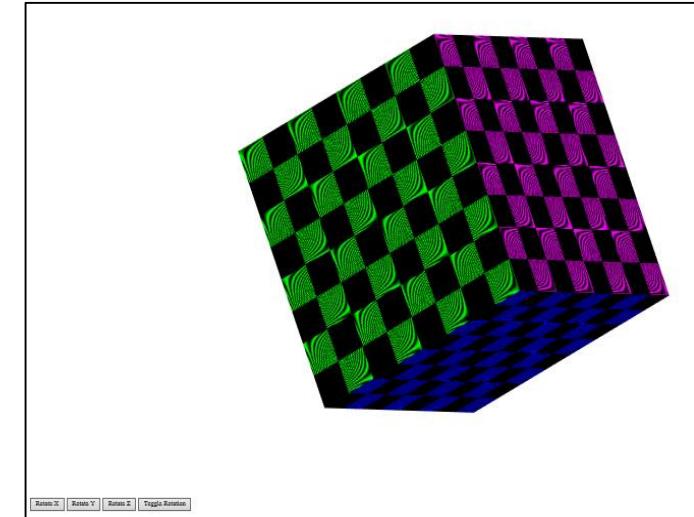
```
var vertices = [  
    vec4( -0.5, -0.5,  0.5, 1.0 ),  
    vec4( -0.5,  0.5,  0.5, 1.0 ),  
    vec4(  0.5,  0.5,  0.5, 1.0 ),  
    vec4(  0.5, -0.5,  0.5, 1.0 ),  
    vec4( -0.5, -0.5, -0.5, 1.0 ),  
    vec4( -0.5,  0.5, -0.5, 1.0 ),  
    vec4(  0.5,  0.5, -0.5, 1.0 ),  
    vec4(  0.5, -0.5,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



textureCubev4.js (6/15)

```
var vertexColors = [  
    vec4( 0.0, 0.0, 0.0, 1.0 ), // black  
    vec4( 1.0, 0.0, 0.0, 1.0 ), // red  
    vec4( 1.0, 1.0, 0.0, 1.0 ), // yellow  
    vec4( 0.0, 1.0, 0.0, 1.0 ), // green  
    vec4( 0.0, 0.0, 1.0, 1.0 ), // blue  
    vec4( 1.0, 0.0, 1.0, 1.0 ), // magenta  
    vec4( 0.0, 1.0, 1.0, 1.0 ), // white  
    vec4( 0.0, 1.0, 1.0, 1.0 ) // cyan  
];  
  
var xAxis = 0;  
var yAxis = 1;  
var zAxis = 2;  
var axis = xAxis;  
  
var theta = [45.0, 45.0, 45.0];
```

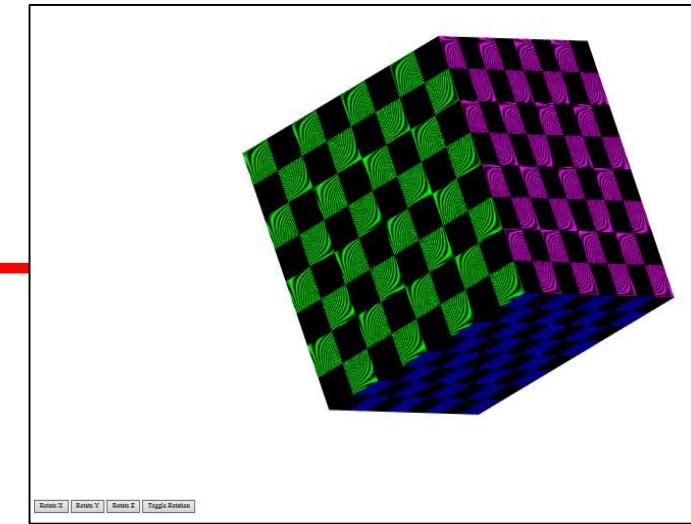
```
var thetaLoc;
```



textureCubev4.js (7/15)

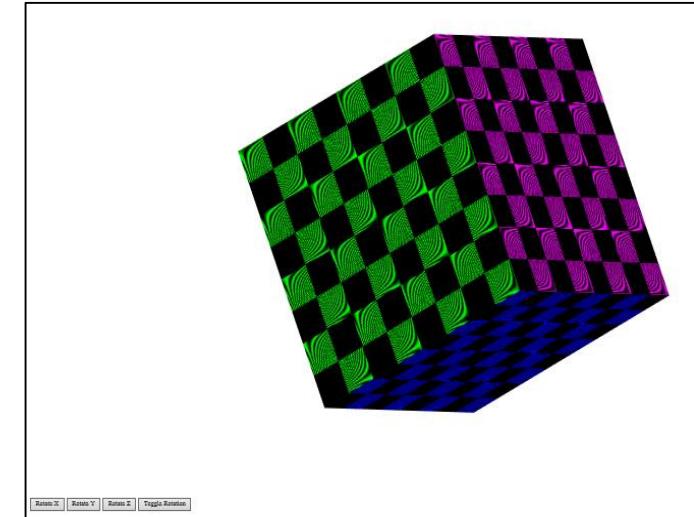
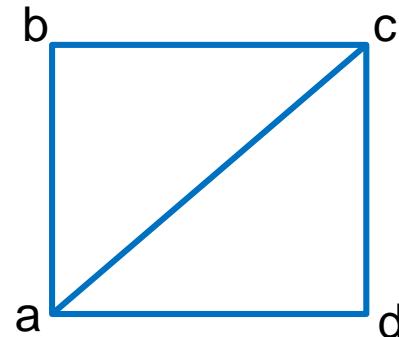
```
function configureTexture() {
    texture1 = gl.createTexture();
    gl.bindTexture( gl.TEXTURE_2D, texture1 );
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texSize, texSize, 0, gl.RGBA, gl.UNSIGNED_BYTE, image1);
    gl.generateMipmap( gl.TEXTURE_2D );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);

    texture2 = gl.createTexture();
    gl.bindTexture( gl.TEXTURE_2D, texture2 );
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texSize, texSize, 0, gl.RGBA, gl.UNSIGNED_BYTE, image2);
    gl.generateMipmap( gl.TEXTURE_2D );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
}
```



textureCubev4.js (8/15)

```
function quad(a, b, c, d) {  
    pointsArray.push(vertices[a]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[0]);  
  
    pointsArray.push(vertices[b]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[1]);  
  
    pointsArray.push(vertices[c]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[2]);
```



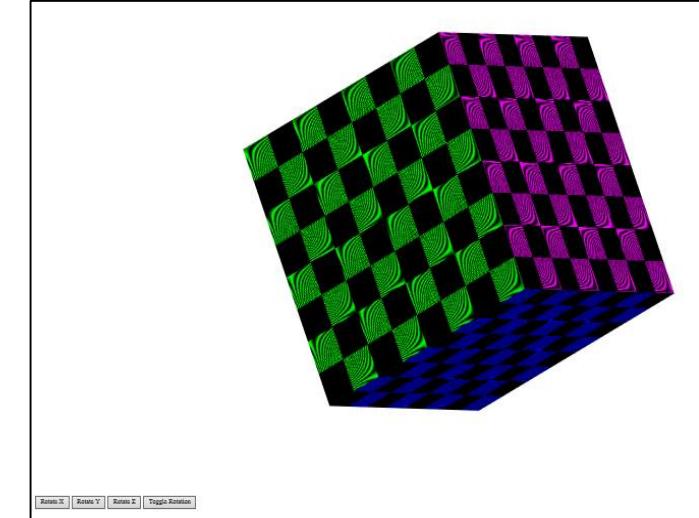
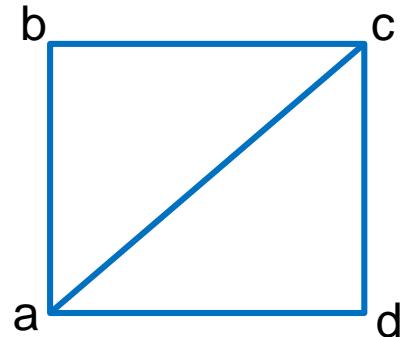
textureCubev4.js (9/15)

```
pointsArray.push(vertices[a]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[0]);

pointsArray.push(vertices[c]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[2]);

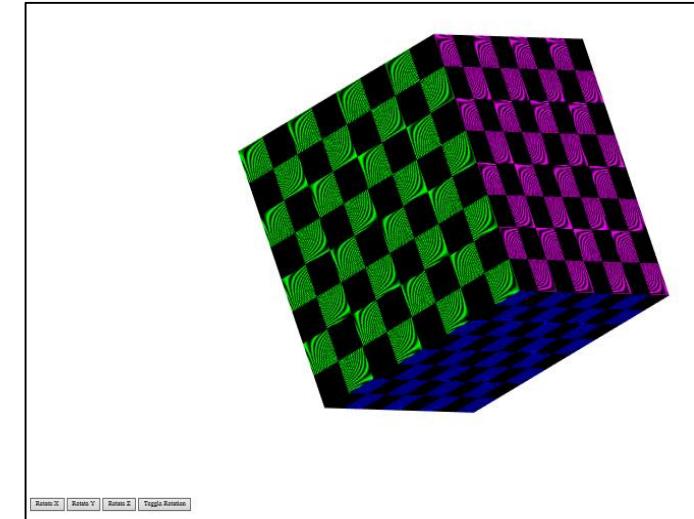
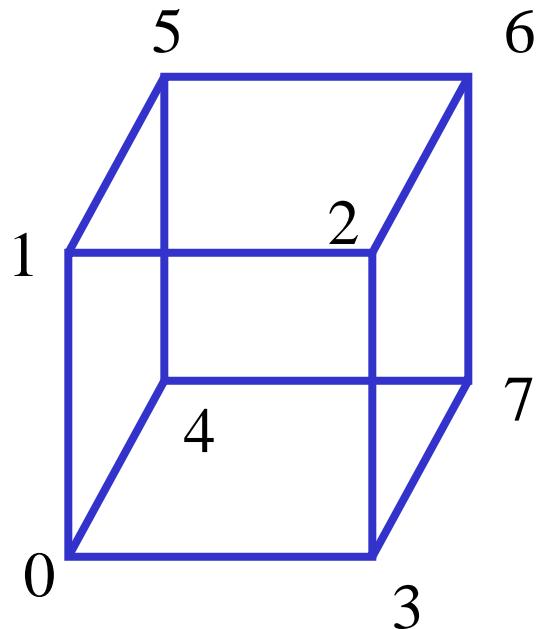
pointsArray.push(vertices[d]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[3]);

} // end of quad(a,b,c,d)
```



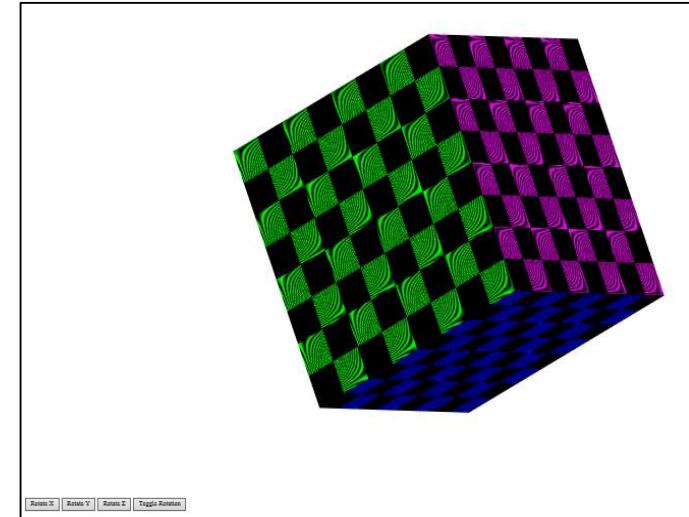
textureCubev4.js (10/15)

```
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 );
}
```



textureCubev4.js (11/15)

```
window.onload = function init() {  
  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );  
  
    gl.enable(gl.DEPTH_TEST);  
  
    // Load shaders and initialize attribute buffers  
  
    program = initShaders( gl, "vertex-shader", "fragment-shader" );  
    gl.useProgram( program );  
  
    colorCube();
```



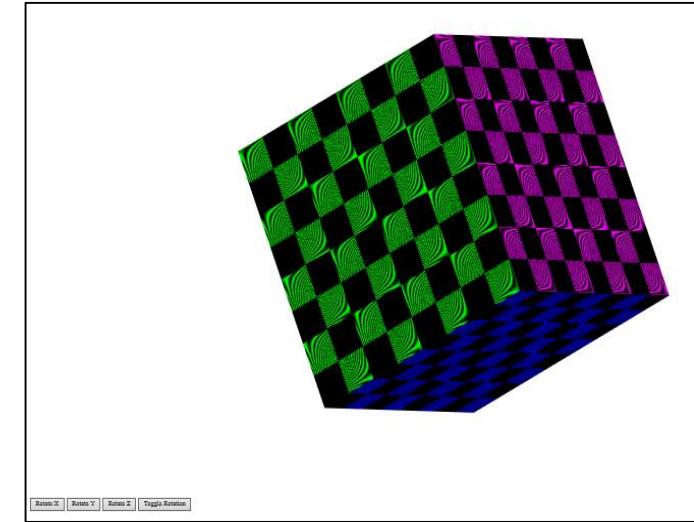
textureCubev4.js (12/15)

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

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



textureCubev4.js (13/15)

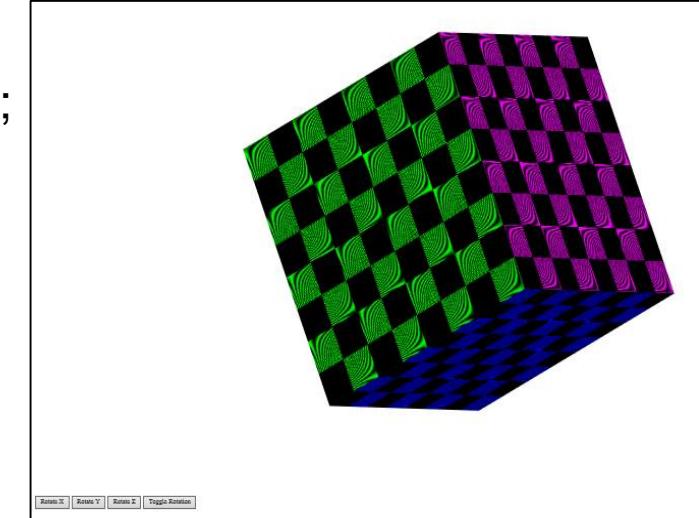
```
var tBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, tBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW );

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

configureTexture();
gl.activeTexture( gl.TEXTURE0 );
gl.bindTexture( gl.TEXTURE_2D, texture1 );
gl.uniform1i(gl.getUniformLocation( program, "Tex0"), 0);

gl.activeTexture( gl.TEXTURE1 );
gl.bindTexture( gl.TEXTURE_2D, texture2 );
gl.uniform1i(gl.getUniformLocation( program, "Tex1"), 1);

thetaLoc = gl.getUniformLocation(program, "theta");
```

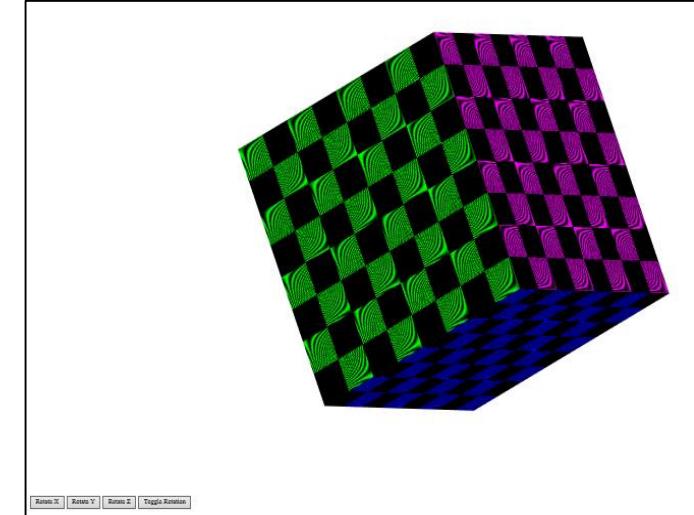


textureCubev4.js (14/15)

```
document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};
document.getElementById("ButtonT").onclick = function() {flag = !flag;};

render();

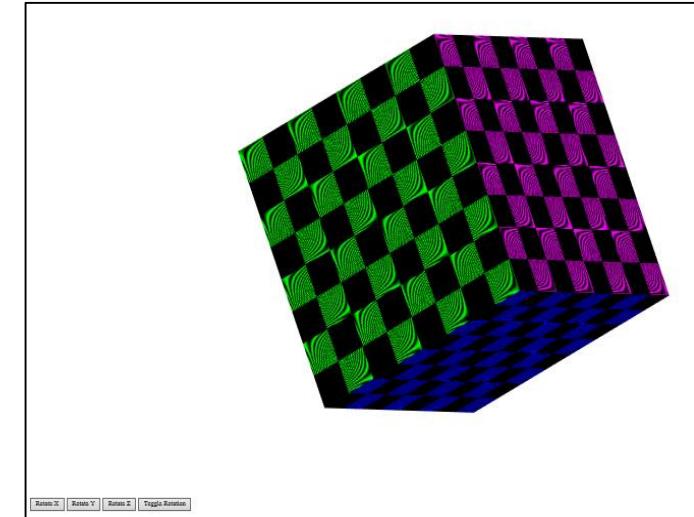
} // end of window.onload
```



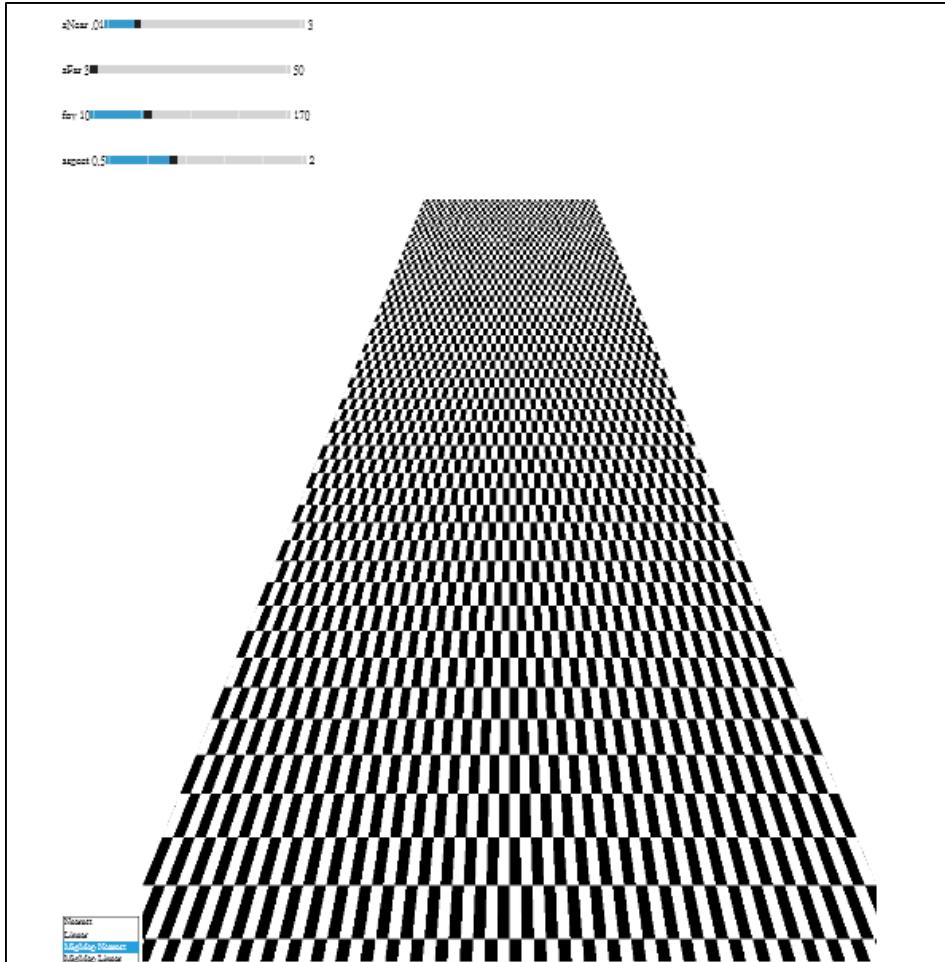
textureCubev4.js (15/15)

```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    if(flag) theta[axis] += 2.0;
    gl.uniform3fv(thetaLoc, theta);
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );
    requestAnimFrame(render);
}

} // end of render()
```



Sample Programs: `textureSquare.html`, `textureSquare.js`

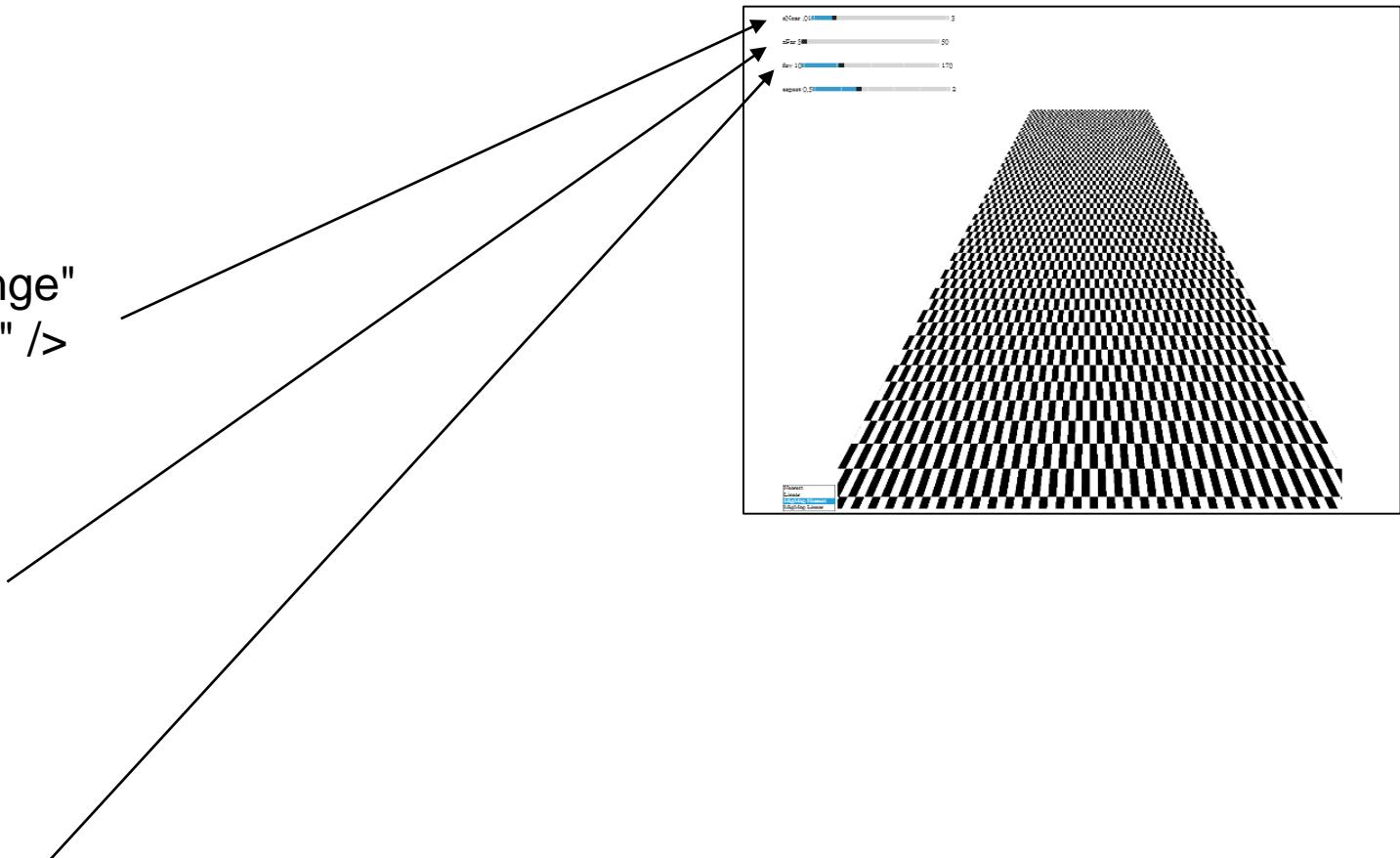


Demo of aliasing with
different texture parameters

textureSquare.html (1/5)

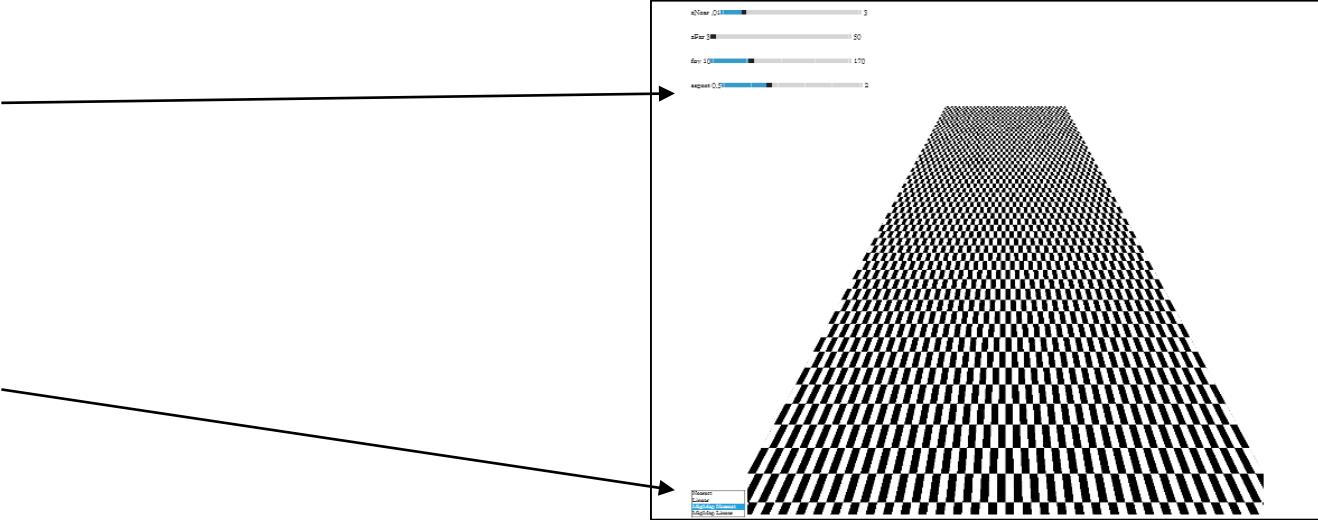
```
<!DOCTYPE html>
<html>

<p> </p>
<div>
zNear .01<input id="zNearSlider" type="range"
min=".01" max="3" step="0.01" value="0.3" />
3
</div>
<div>
zFar 3<input id="zFarSlider" type="range"
min="3" max="50" step="1.0" value="3" />
50
</div>
<div>
fov 10<input id="fovSlider" type="range"
min="10" max="170" step="5" value="45" />
170
</div>
```



textureSquare.html (2/5)

```
<div>  
aspect 0.5<input id="aspectSlider" type="range"  
min="0.5" max="2" step="0.1" value="1" />  
2  
</div>  
<select id="Texture Style" size="4">  
  <option value="0">Nearest</option>  
  <option value="1">Linear</option>  
  <option value="2">MipMap Nearest</option>  
  <option value="3">MipMap Linear</option>  
</select>
```

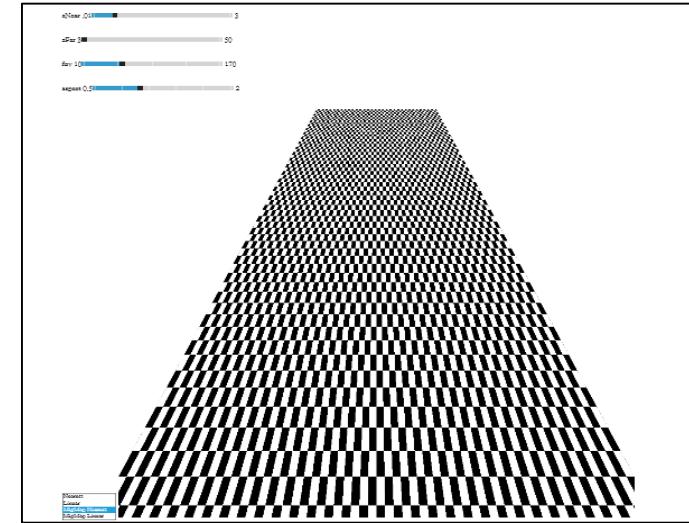


textureSquare.html (3/5)

```
<script id="vertex-shader" type="x-shader/x-vertex">

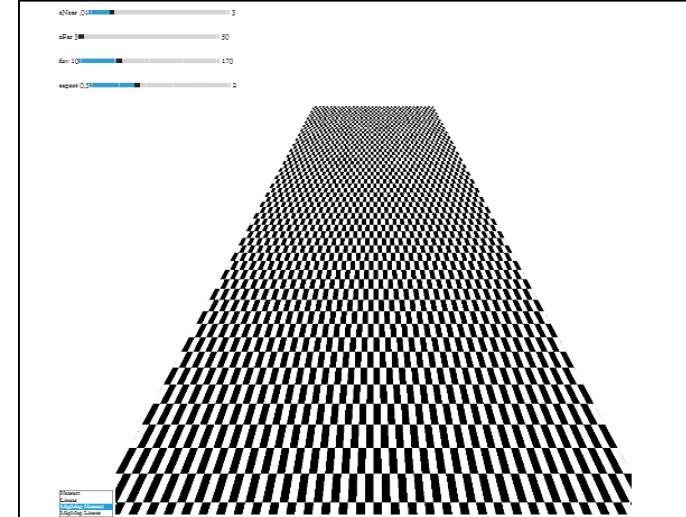
attribute vec4 vPosition;
attribute vec4 vColor;
attribute vec2 vTexCoord;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
varying vec4 fColor;
varying vec2 fTexCoord;
uniform vec3 theta;

void main()
{
    fColor = vColor;
    fTexCoord = vTexCoord;
    gl_Position = projectionMatrix*modelViewMatrix*vPosition;
}
</script>
```



textureSquare.html (4/5)

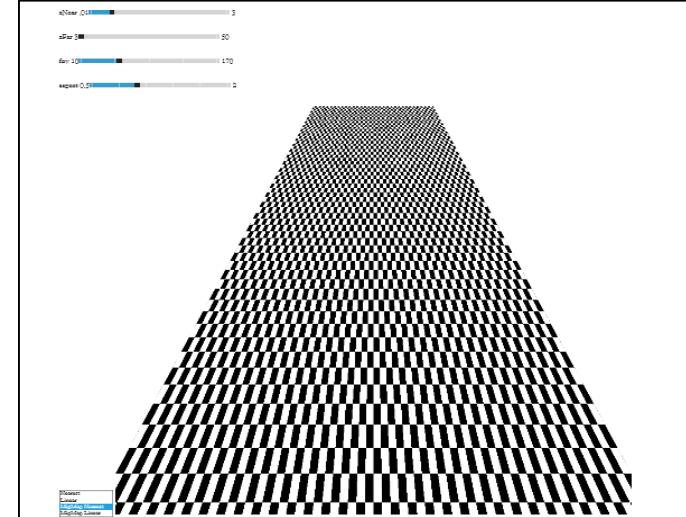
```
<script id="fragment-shader" type="x-shader/x-fragment">  
  
precision mediump float;  
  
varying vec4 fColor;  
varying vec2 fTexCoord;  
  
uniform sampler2D texture;  
  
void  
main()  
{  
    gl_FragColor = fColor * texture2D( texture, fTexCoord );  
}  
</script>
```



textureSquare.html (5/5)

```
<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="textureSquare.js"></script>
```

```
<body>
<canvas id="gl-canvas" width="1024" height="1024">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



textureSquare.js (1/14)

```
var canvas;
var gl;

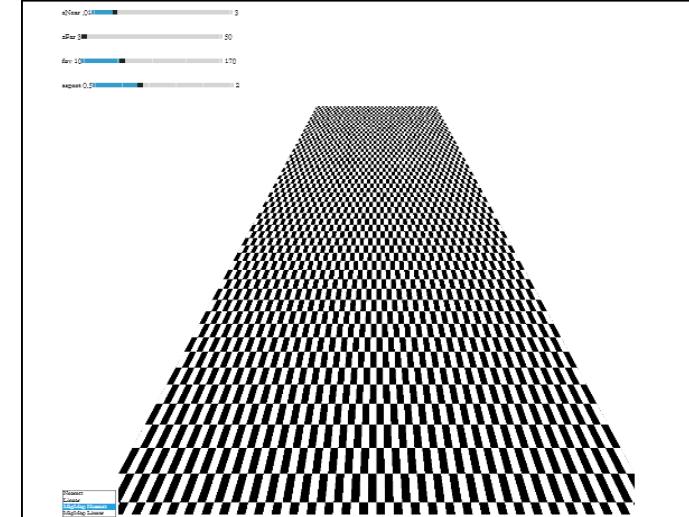
var numVertices = 6;

var texSize = 256;
var numChecks = 64;

//var flag = true;

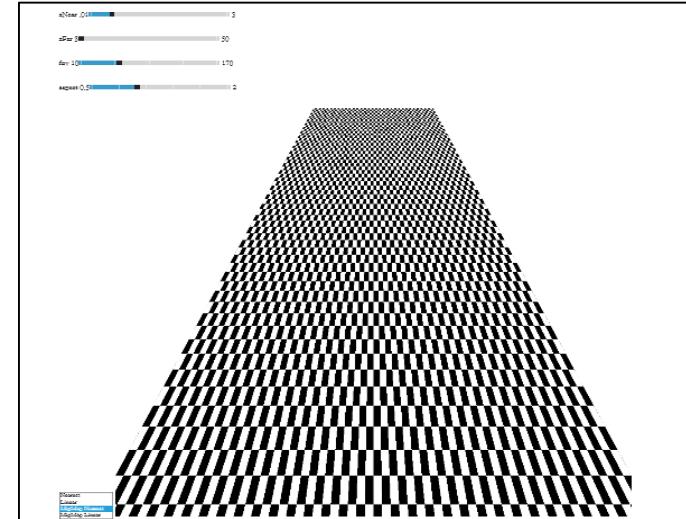
var near = 0.3;
var far = 3.0;

var fovy = 45.0; // Field-of-view in Y direction angle (in degrees)
var aspect = 1.0; // Viewport aspect ratio
```



textureSquare.js (2/14)

```
var texture1, texture2, texture3, texture4;  
  
// Create a checkerboard pattern using floats  
  
var image1 = new Uint8Array(4*texSize*texSize);  
for ( var i = 0; i < texSize; i++ ) {  
    for ( var j = 0; j < texSize; j++ ) {  
        var patchx = Math.floor( i / (texSize/numChecks) );  
        var patchy = Math.floor( j / (texSize/numChecks) );  
        if(patchx%2 ^ patchy%2) c = 255;  
        else c = 0;  
        //c = 255*((i & 0x8) == 0) ^ ((j & 0x8) == 0))  
        image1[4*i*texSize+4*j] = c;  
        image1[4*i*texSize+4*j+1] = c;  
        image1[4*i*texSize+4*j+2] = c;  
        image1[4*i*texSize+4*j+3] = 255;  
    }  
}
```

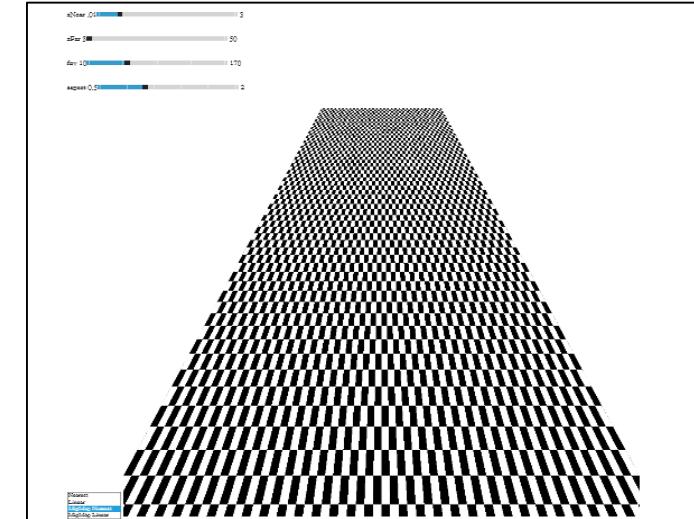
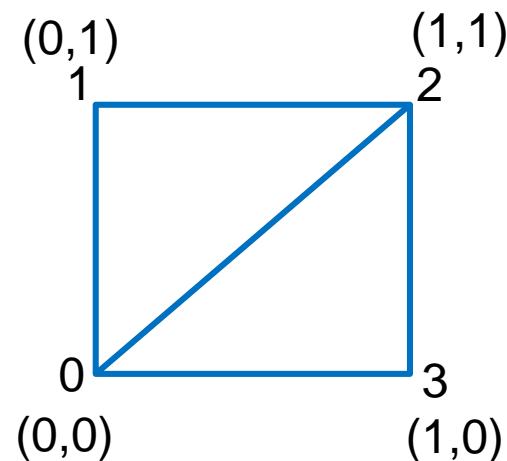


textureSquare.js (3/14)

```
var pointsArray = [];
var colorsArray = [];
var texCoordsArray = [];

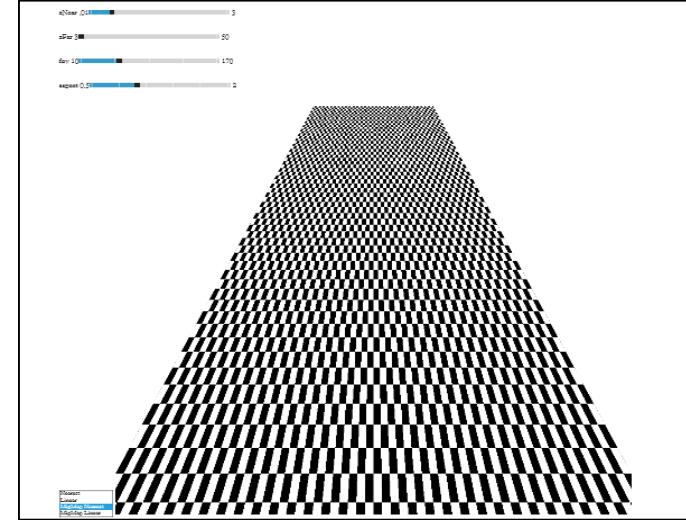
var texCoord = [
    vec2(0, 0),
    vec2(0, 1),
    vec2(1, 1),
    vec2(1, 0)
];
var zmin = 1.0;
var zmax = 50.0;
```

```
var vertices = [
    vec4( -5.5, 0.0, zmax, 1.0 ),
    vec4( -5.5, 0.0, zmin, 1.0 ),
    vec4( 5.5, 0.0, zmin, 1.0 ),
    vec4( 5.5, 0.0, zmax, 1.0 ),
];
```



textureSquare.js (4/14)

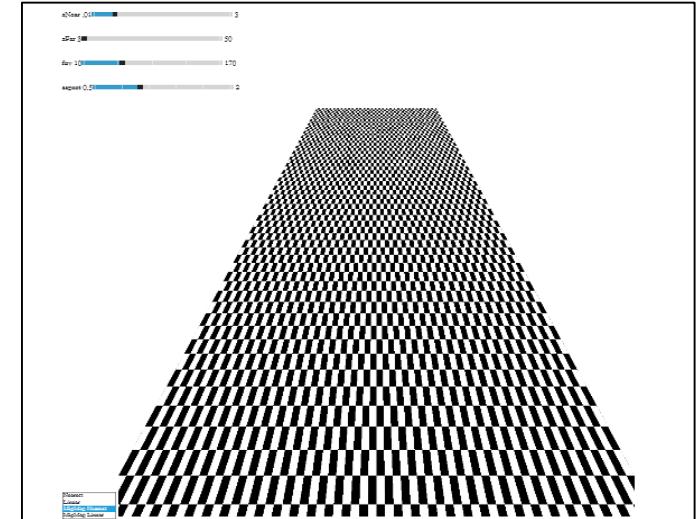
```
var modelViewMatrix, projectionMatrix;  
var modelViewMatrixLoc, projectionMatrixLoc;  
var eye = vec3(0.0, 10.0, 0.0);  
var at = vec3(0.0, 0.0, (zmax+zmin)/4);  
const up = vec3(0.0, 1.0, 0.0);  
  
var vertexColors = [  
    vec4( 0.0, 0.0, 0.0, 1.0 ), // black  
    vec4( 1.0, 1.0, 1.0, 1.0 ), // white  
    vec4( 1.0, 0.0, 0.0, 1.0 ), // red  
    vec4( 1.0, 1.0, 0.0, 1.0 ), // yellow  
    vec4( 0.0, 1.0, 0.0, 1.0 ), // green  
    vec4( 0.0, 0.0, 1.0, 1.0 ), // blue  
    vec4( 1.0, 0.0, 1.0, 1.0 ), // magenta  
    vec4( 0.0, 1.0, 1.0, 1.0 ) // cyan  
];
```



textureSquare.js (5/14)

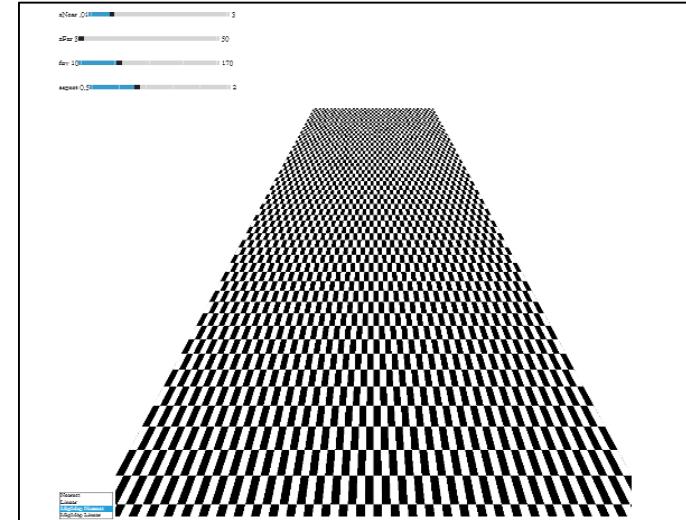
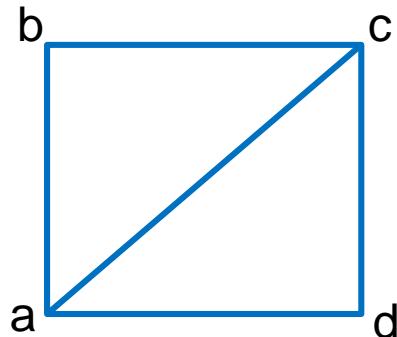
```
window.onload = init;

function configureTexture(image) {
    texture1 = gl.createTexture();
    gl.bindTexture( gl.TEXTURE_2D, texture1 );
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texSize, texSize, 0, gl.RGBA, gl.UNSIGNED_BYTE, image);
    gl.generateMipmap( gl.TEXTURE_2D );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );
};
```



textureSquare.js (6/14)

```
function quad(a, b, c, d) {  
  
    pointsArray.push(vertices[a]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[0]);  
  
    pointsArray.push(vertices[b]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[1]);  
  
    pointsArray.push(vertices[c]);  
    colorsArray.push(vertexColors[a]);  
    texCoordsArray.push(texCoord[2]);
```



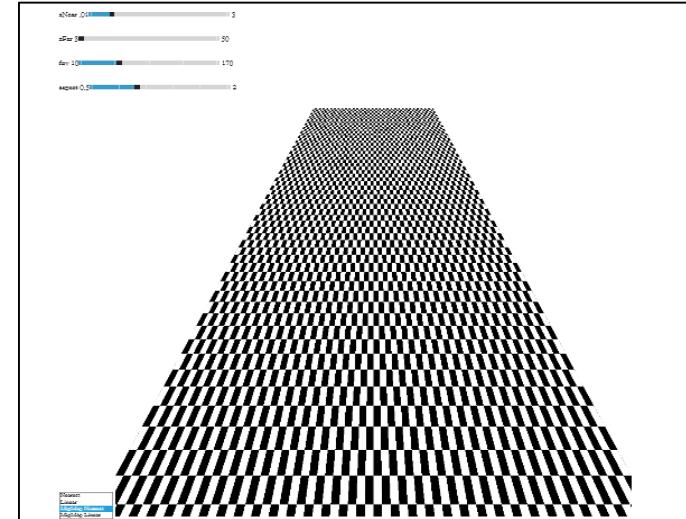
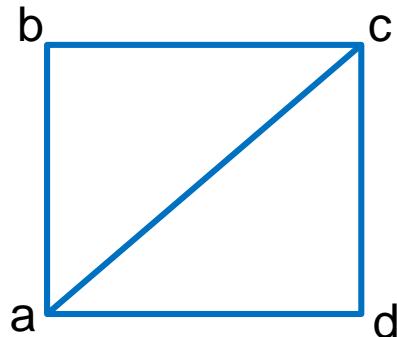
textureSquare.js (7/14)

```
pointsArray.push(vertices[a]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[0]);

pointsArray.push(vertices[c]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[2]);

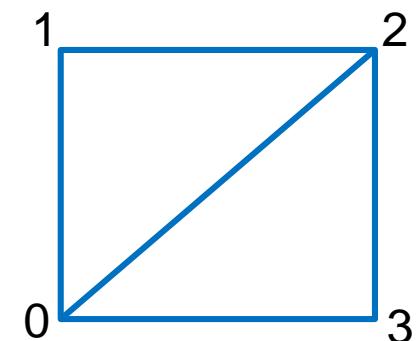
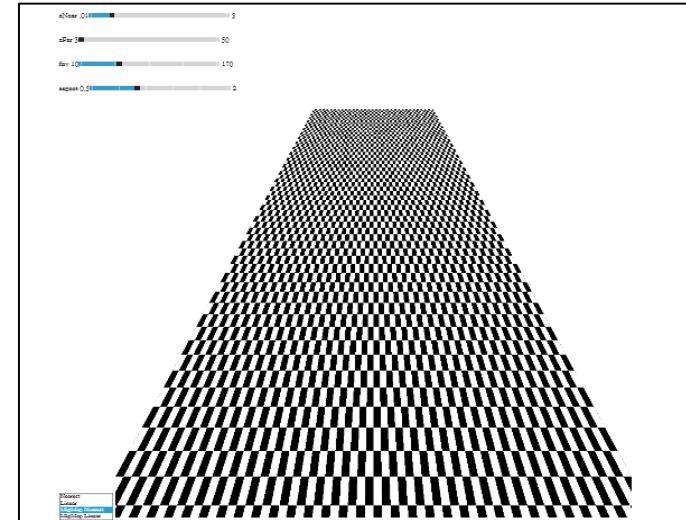
pointsArray.push(vertices[d]);
colorsArray.push(vertexColors[a]);
texCoordsArray.push(texCoord[3]);

} // end of quad(a,b,c,d)
```



textureSquare.js (8/14)

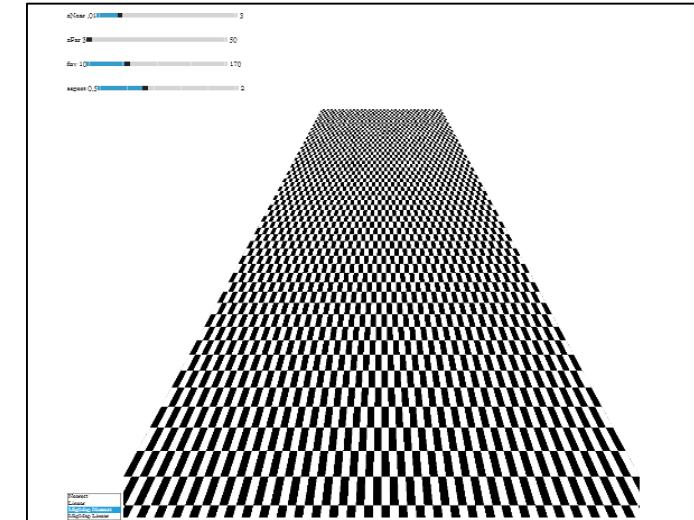
```
function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 0.5, 0.5, 0.5, 1.0 );  
  
    //  
    // Load shaders and initialize attribute buffers  
    //  
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );  
    gl.useProgram( program );  
  
    quad( 1, 0, 3, 2 );
```



textureSquare.js (9/14)

```
var cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(colorsArray), 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(pointsArray), gl.STATIC_DRAW);
var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```

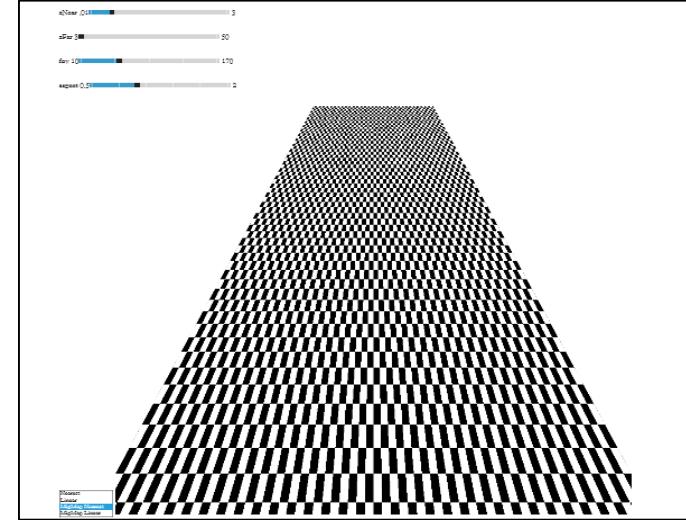


textureSquare.js (10/14)

```
var tBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, tBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW );
var vTexCoord = gl.getAttribLocation( program, "vTexCoord");
gl.vertexAttribPointer(vTexCoord, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vTexCoord);

configureTexture(image1);

modelViewMatrixLoc = gl.getUniformLocation( program, "modelViewMatrix" );
projectionMatrixLoc = gl.getUniformLocation( program, "projectionMatrix" );
```



textureSquare.js (11/14)

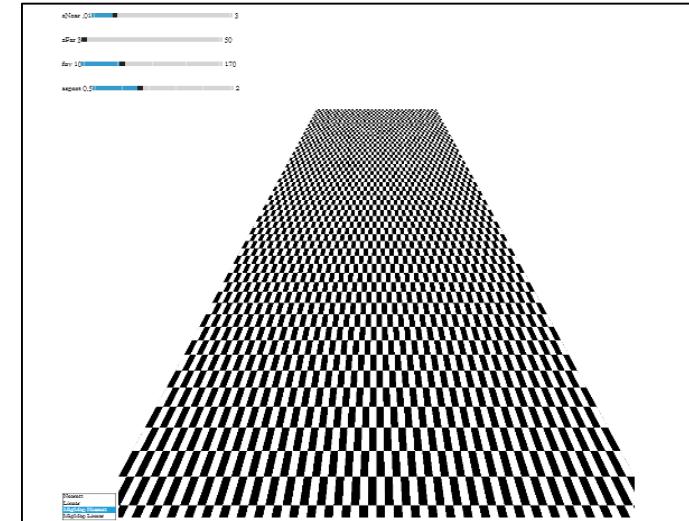
```
// sliders for viewing parameters

document.getElementById("zFarSlider").onchange = function() {
    far = event.srcElement.value;
};

document.getElementById("zNearSlider").onchange = function() {
    near = event.srcElement.value;
};

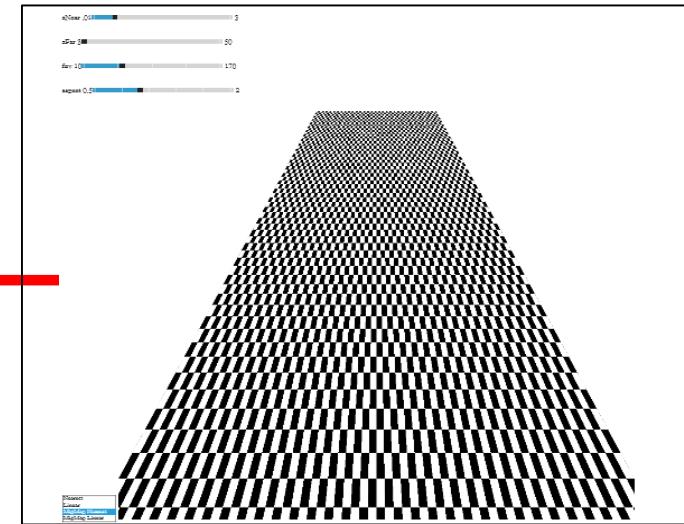
document.getElementById("aspectSlider").onchange = function() {
    aspect = event.srcElement.value;
};

document.getElementById("fovSlider").onchange = function() {
    fovy = event.srcElement.value;
};
```



textureSquare.js (12/14)

```
document.getElementById("Texture Style").onclick = function(event) {
    switch(event.srcElement.index) {
        case 0: // Nearest
            gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_NEAREST);
            gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );
            break;
        case 1: // Linear
            gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR);
            gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR );
            break;
    }
}
```

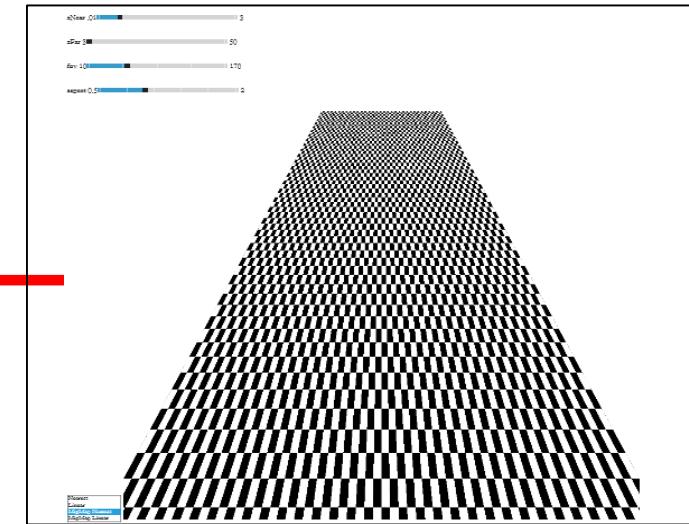


textureSquare.js (13/14)

```
case 2: // MipMap Nearest
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );
    break;
case 3: // MipMap Linear
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR );
    break;
}
};

render();

} // end of init()
```

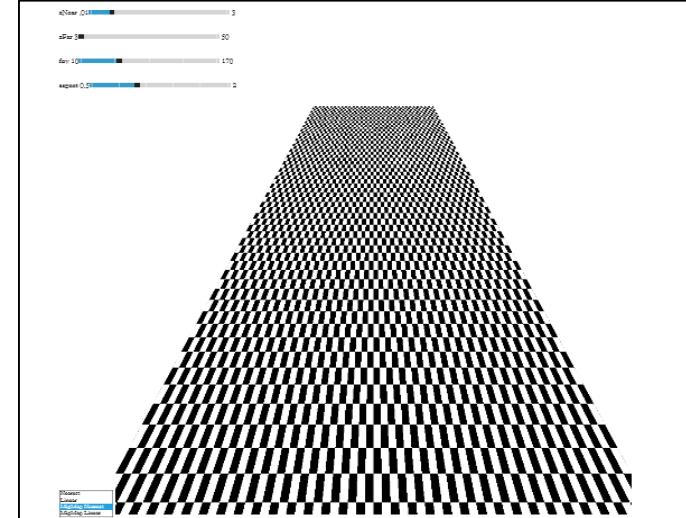


textureSquare.js (14/14)

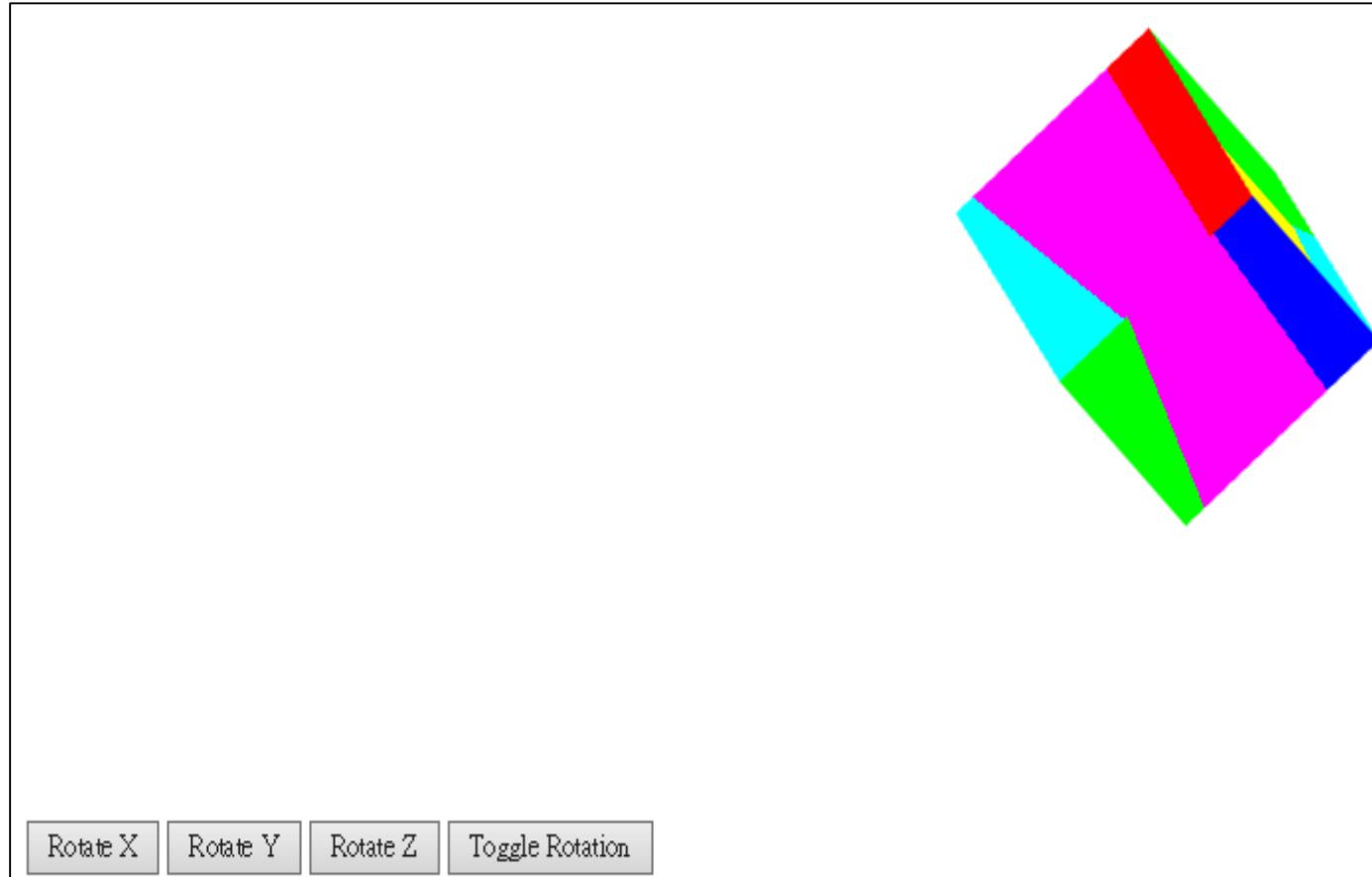
```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT);

    modelViewMatrix = lookAt(eye, at , up);
    projectionMatrix = perspective(fovy, aspect, near, far);

    gl.uniformMatrix4fv( modelViewMatrixLoc, false, flatten(modelViewMatrix) );
    gl.uniformMatrix4fv( projectionMatrixLoc, false, flatten(projectionMatrix) );
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );
    requestAnimFrame(render);
} // end of render()
```



Sample Programs: reflectionMap.html, reflectionMap.js



Reflection map of a colored cube onto another cube rotating inside of it

reflectionMap.html (1/5)

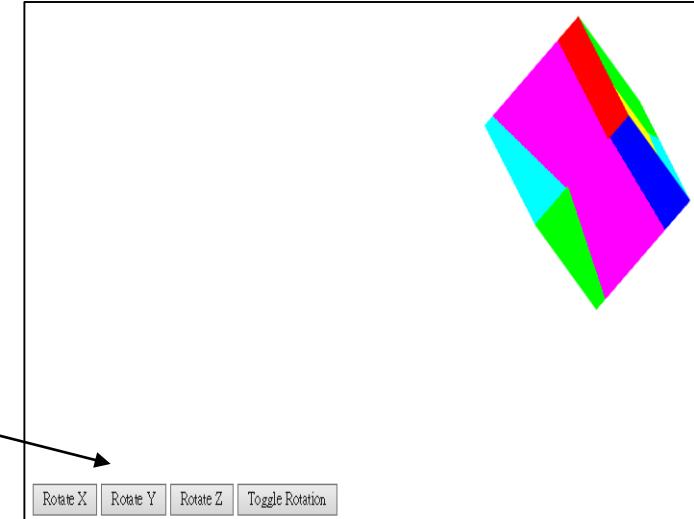
```
<!DOCTYPE html>
<html>

<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
<button id = "ButtonT">Toggle Rotation</button>

<script id="vertex-shader" type="x-shader/x-vertex">

varying vec3 R;
attribute vec4 vPosition;
attribute vec4 vNormal;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec3 theta;
```

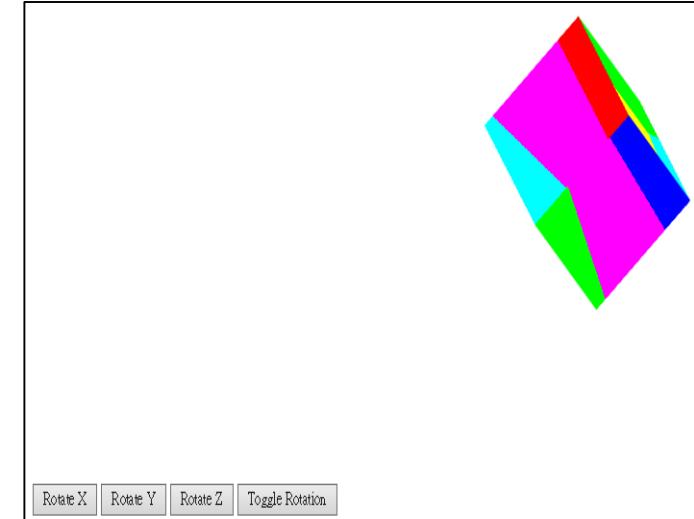


reflectionMap.html (2/5)

```
void main()
{
    vec3 angles = radians( theta );
    vec3 c = cos( angles );
    vec3 s = sin( angles );

    // Remember: these matrices are column-major
    mat4 rx = mat4( 1.0, 0.0, 0.0, 0.0,
                    0.0, c.x, s.x, 0.0,
                    0.0, -s.x, c.x, 0.0,
                    0.0, 0.0, 0.0, 1.0 );

    mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                    0.0, 1.0, 0.0, 0.0,
                    s.y, 0.0, c.y, 0.0,
                    0.0, 0.0, 0.0, 1.0 );
```



reflectionMap.html (3/5)

```
mat4 rz = mat4( c.z, -s.z, 0.0, 0.0,
                 s.z, c.z, 0.0, 0.0,
                 0.0, 0.0, 1.0, 0.0,
                 0.0, 0.0, 0.0, 1.0 );

mat4 ModelViewMatrix = modelViewMatrix*rz*ry*rx;
gl_Position = projectionMatrix*ModelViewMatrix*vPosition;

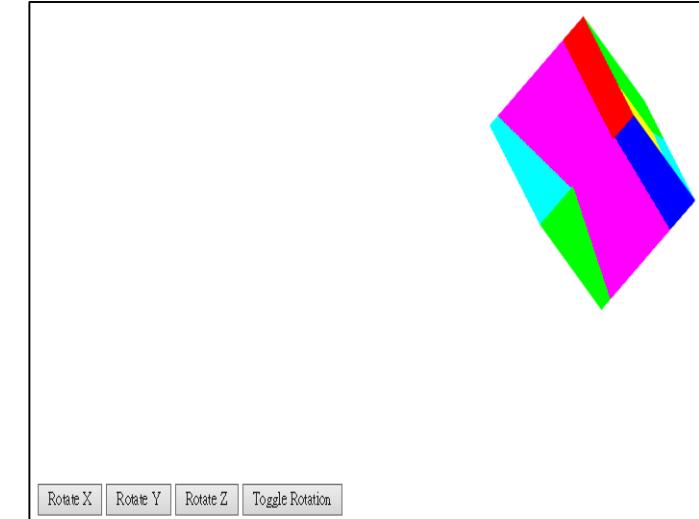
// try swapping the line below for one above

//gl_Position = vPosition;
vec4 eyePos = ModelViewMatrix*vPosition;

vec4 N = ModelViewMatrix*vNormal;
R = reflect(eyePos.xyz, N.xyz);

}

</script>
```



reflectionMap.html (4/5)

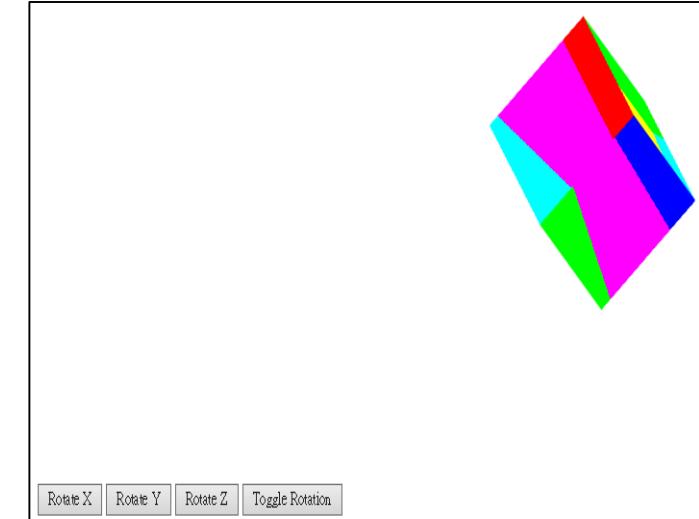
```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

varying vec3 R;
uniform samplerCube texMap;

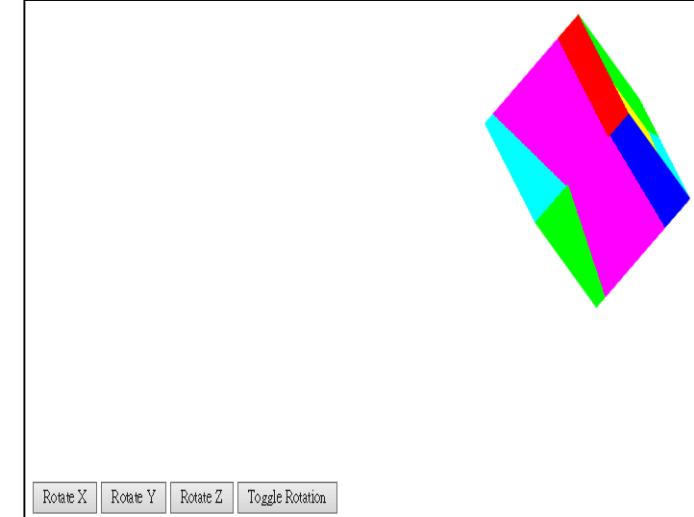
void main()
{
    vec4 texColor = textureCube(texMap, R);
    gl_FragColor = texColor;
}
</script>
```

```
<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="reflectionMap.js"></script>
```



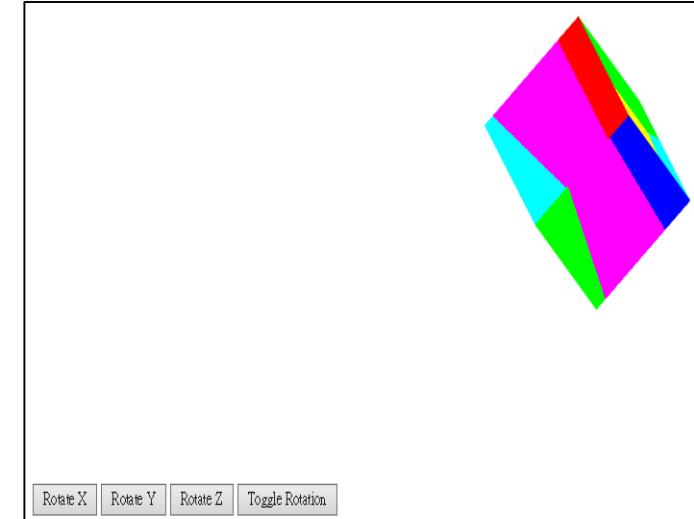
reflectionMap.html (5/5)

```
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



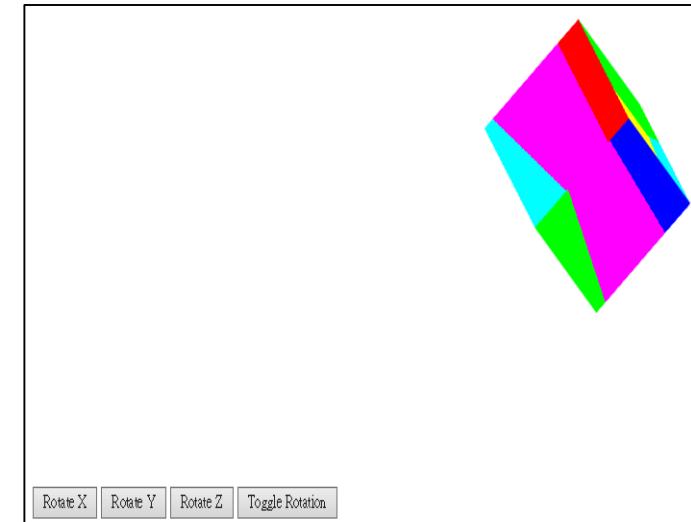
reflectionMap.js (1/13)

```
var canvas;  
var gl;  
  
var numVertices = 36;  
  
var texSize = 4;  
var numChecks = 2;  
  
var flag = true;  
  
var image2 = new Uint8Array(4*texSize*texSize);  
  
var red      = new Uint8Array([255, 0, 0, 255]);  
var green    = new Uint8Array([ 0, 255, 0, 255]);  
var blue     = new Uint8Array([ 0, 0, 255, 255]);  
var cyan     = new Uint8Array([ 0, 255, 255, 255]);  
var magenta  = new Uint8Array([255, 0, 255, 255]);  
var yellow   = new Uint8Array([255, 255, 0, 255]);
```



reflectionMap.js (2/13)

```
var cubeMap;  
  
var pointsArray = [];  
var normalsArray = [];  
  
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 )  
];
```



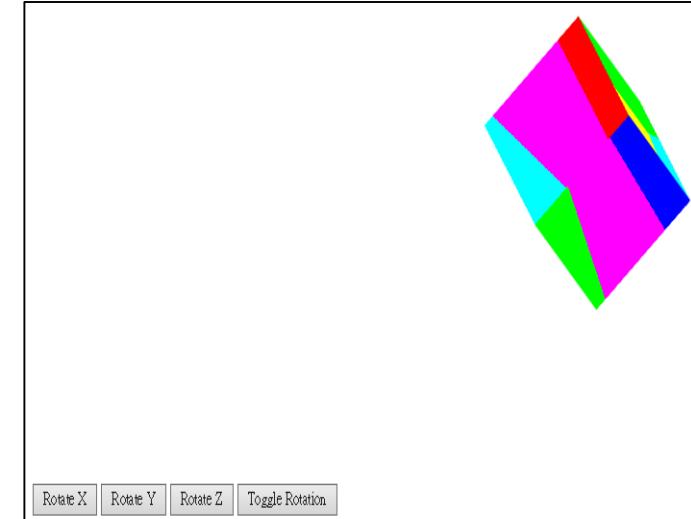
reflectionMap.js (3/13)

```
window.onload = init;
```

```
var xAxis = 0;  
var yAxis = 1;  
var zAxis = 2;  
var axis = xAxis;
```

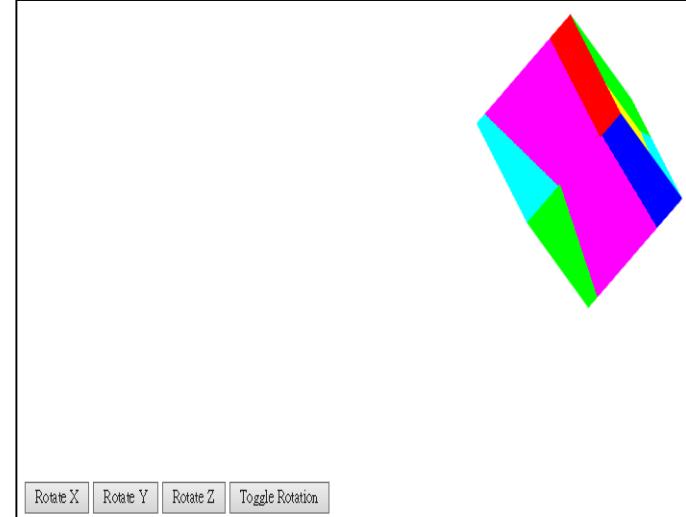
```
var theta = [45.0, 45.0, 45.0];
```

```
var thetaLoc;
```



reflectionMap.js (4/13)

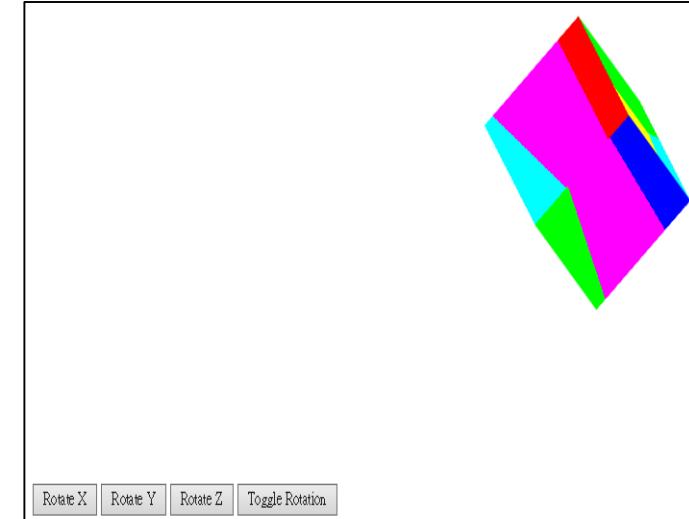
```
function configureCubeMap() {  
  
    for ( var i = 0; i < texSize; i++ ) {  
        for ( var j = 0; j < texSize; j++ ) {  
            var c;  
            var patchx = Math.floor( i / (texSize/numChecks) );  
            var patchy = Math.floor( j / (texSize/numChecks) );  
            if(patchx%2 != patchy%2) c = 255;  
            else c = 0;  
            image2[4*i*texSize+4*j] = c;  
            image2[4*i*texSize+4*j+1] = c;  
            image2[4*i*texSize+4*j+2] = c;  
            image2[4*i*texSize+4*j+3] = 255;  
        }  
    }  
}
```



reflectionMap.js (5/13)

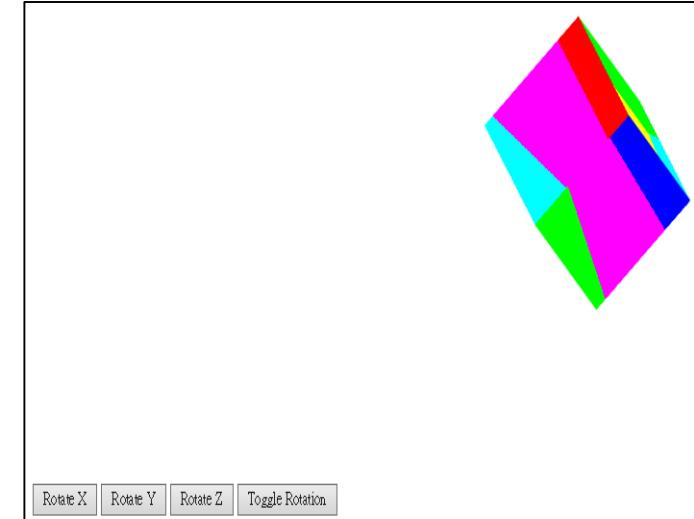
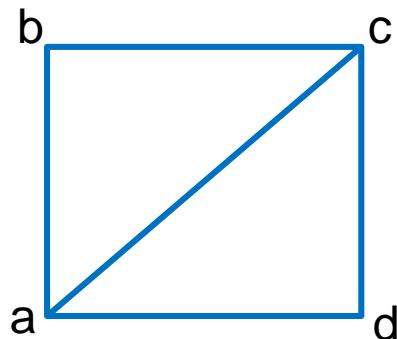
```
cubeMap = gl.createTexture();
gl.bindTexture(gl.TEXTURE_CUBE_MAP, cubeMap);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_X, 0, gl.RGBA,
    1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, red);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_X, 0, gl.RGBA,
    1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, green);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_Y, 0, gl.RGBA,
    1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, blue);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_Y, 0, gl.RGBA,
    1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, cyan);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_Z, 0, gl.RGBA,
    1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, yellow);
gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_Z, 0, gl.RGBA,
    1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE, magenta);

gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
} // end of configureCubeMap()
```



reflectionMap.js (6/13)

```
function quad(a, b, c, d) {  
  
    var t1 = subtract(vertices[b], vertices[a]);  
    var t2 = subtract(vertices[c], vertices[b]);  
    var normal = cross(t1, t2);  
    normal = normalize(normal);  
  
    pointsArray.push(vertices[a]);  
    normalsArray.push(normal);  
  
    pointsArray.push(vertices[b]);  
    normalsArray.push(normal);  
  
    pointsArray.push(vertices[c]);  
    normalsArray.push(normal );
```



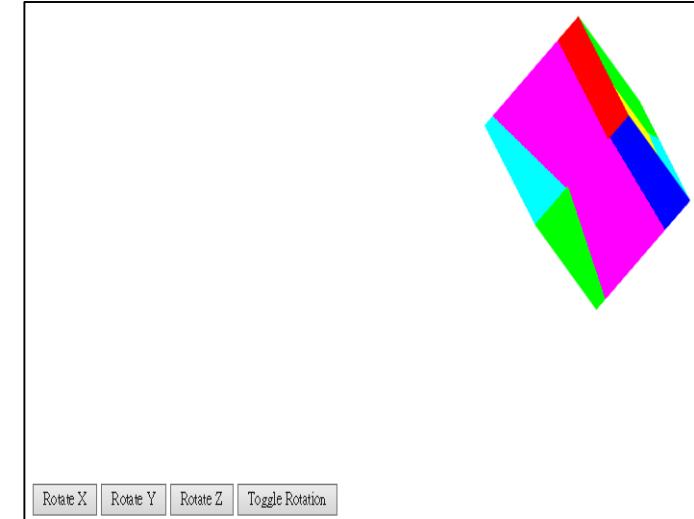
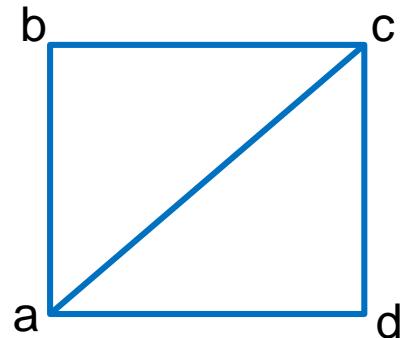
reflectionMap.js (7/13)

```
pointsArray.push(vertices[a]);
normalsArray.push(normal);

pointsArray.push(vertices[c]);
normalsArray.push( normal );

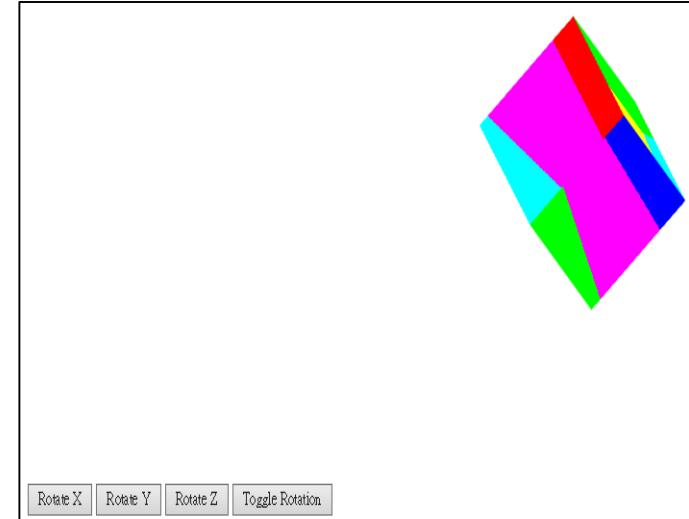
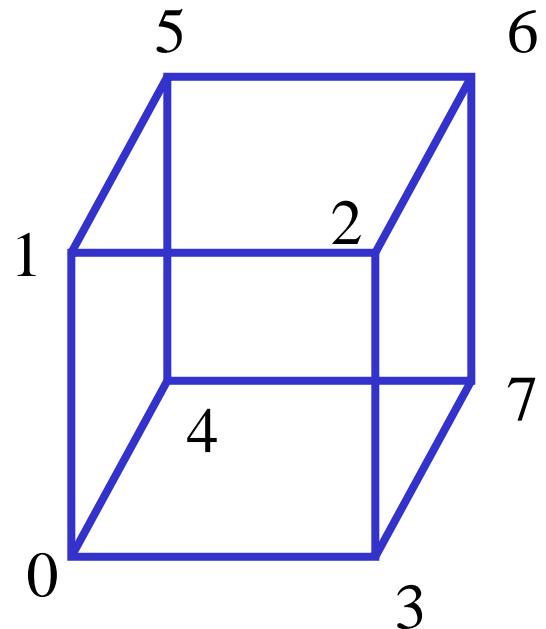
pointsArray.push(vertices[d]);
normalsArray.push(normal);

} // end of quad(a,b,c,d)
```



reflectionMap.js (8/13)

```
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 );
}
```



reflectionMap.js (9/13)

```
function init() {
    canvas = document.getElementById( "gl-canvas" );

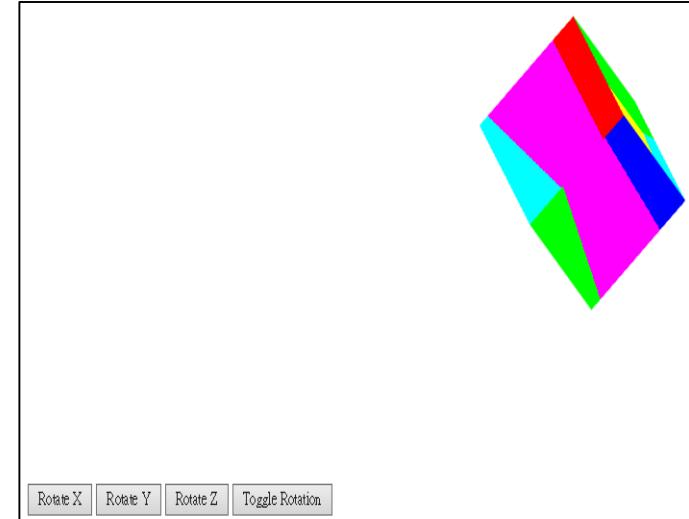
    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

    gl.enable(gl.DEPTH_TEST);

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

    colorCube();
```



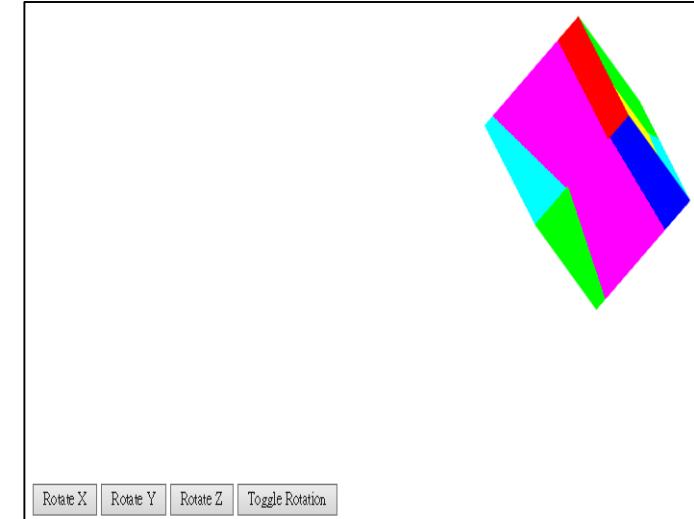
reflectionMap.js (10/13)

```
var nBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, nBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW );

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

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

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



reflectionMap.js (11/13)

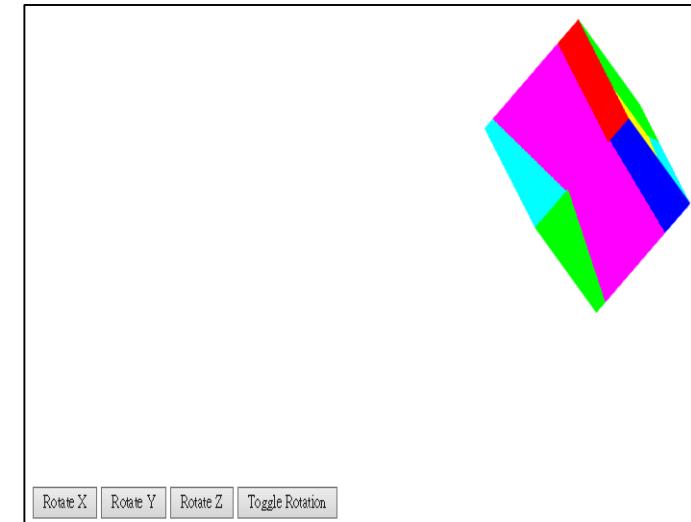
```
var eye = vec3(0.0, 0.0, 1.0);
var at   = vec3(0.0, 0.0, 0.0);
var up   = vec3(0.0, 1.0, 0.0);

var modelViewMatrixLoc = gl.getUniformLocation( program, "modelViewMatrix" );
var modelViewMatrix = lookAt(eye, at, up);
gl.uniformMatrix4fv( modelViewMatrixLoc, false, flatten(modelViewMatrix) );

var projectionMatrixLoc = gl.getUniformLocation( program, "projectionMatrix" );
var projectionMatrix = ortho(-2, 2, -2, 2, -10, 10);
gl.uniformMatrix4fv( projectionMatrixLoc, false, flatten(projectionMatrix) );

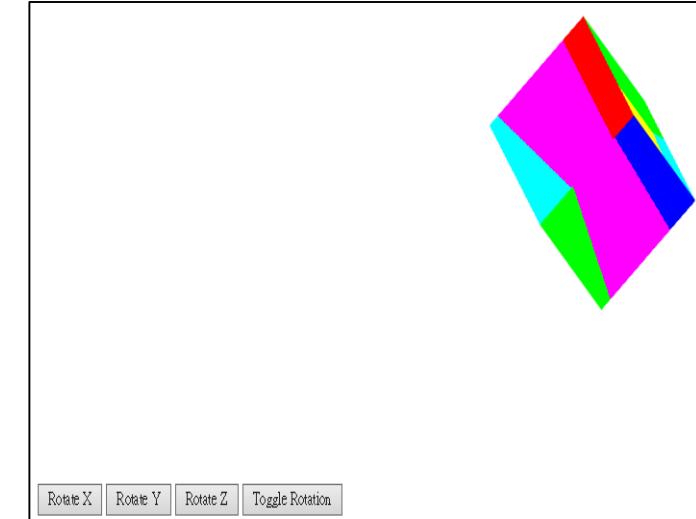
configureCubeMap();
gl.activeTexture( gl.TEXTURE0 );
gl.uniform1i(gl.getUniformLocation(program, "texMap"),0);

thetaLoc = gl.getUniformLocation(program, "theta");
```



reflectionMap.js (12/13)

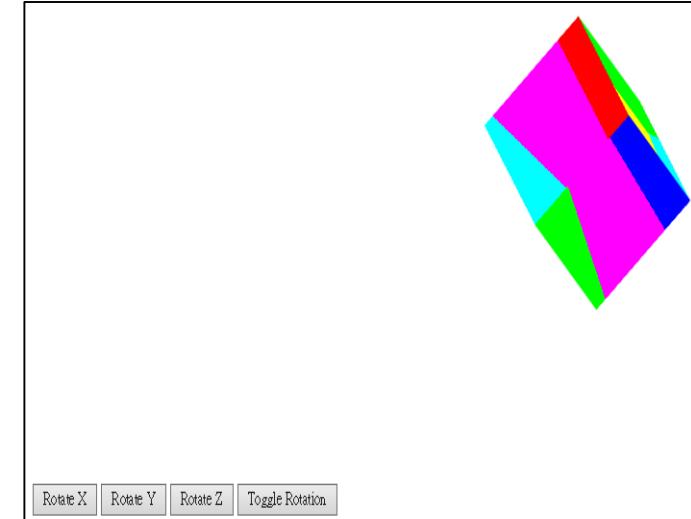
```
document.getElementById("ButtonX").onclick = function() {axis = xAxis;}  
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};  
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};  
document.getElementById("ButtonT").onclick = function() {flag = !flag;};  
  
render();  
  
} // end of init()
```



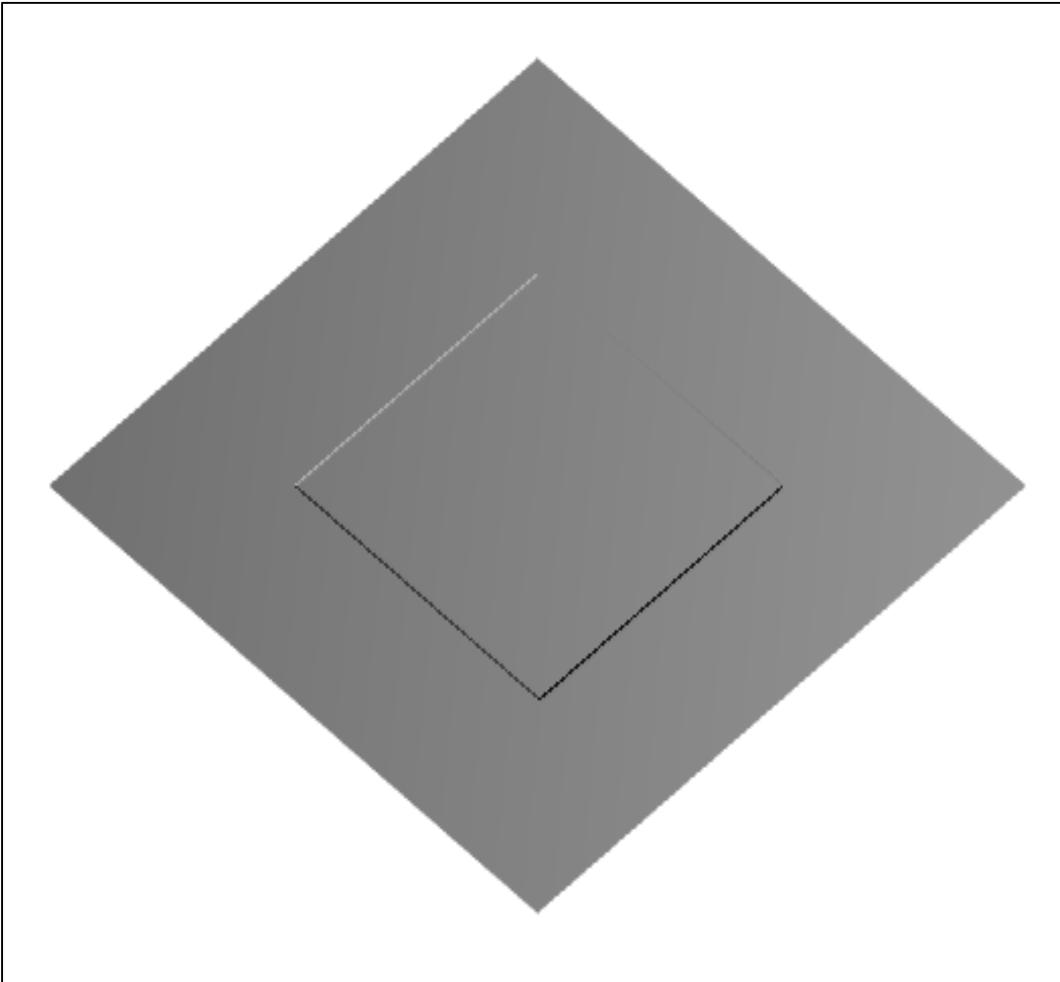
reflectionMap.js (13/13)

```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    if(flag) theta[axis] += 2.0;
    gl.uniform3fv(thetaLoc, theta);
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );
    requestAnimFrame(render);
}

} // end of render()
```



Sample Programs: bumpMap.html, bumpMap.js



Bump map of a single square with a square "bump" in the middle and a rotating light

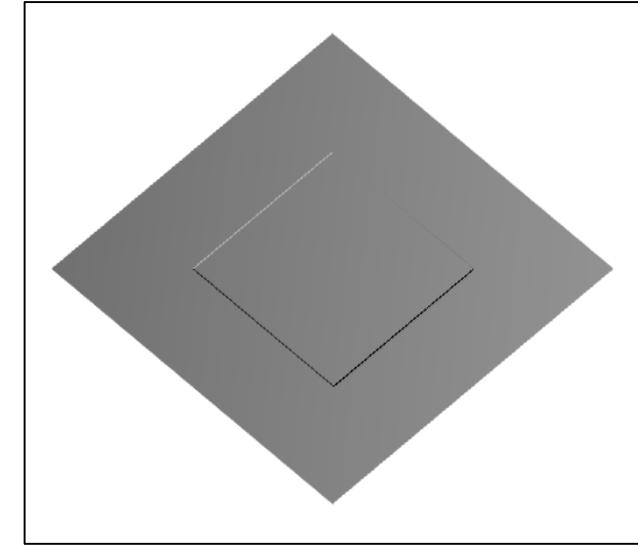
bumpMap.html (1/5)

```
<!DOCTYPE html>
<html>
<script id="vertex-shader" type="x-shader/x-vertex">

/* bump map vertex shader */
varying vec3 L; /* light vector in texture-space coordinates */
varying vec3 V; /* view vector in texture-space coordinates */

attribute vec2 vTexCoord;
attribute vec4 vPosition;

uniform vec4 normal;
uniform vec4 lightPosition;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform mat3 normalMatrix;
uniform vec3 objTangent; /* tangent vector in object coordinates */
varying vec2 fTexCoord;
```



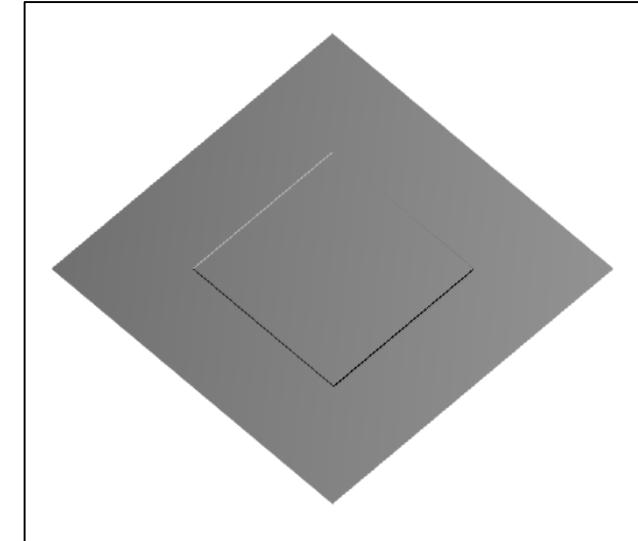
bumpMap.html (2/5)

```
void main()
{
    gl_Position = projectionMatrix*modelViewMatrix*vPosition;
    fTexCoord = vTexCoord;

    vec3 eyePosition = (modelViewMatrix*vPosition).xyz;
    vec3 eyeLightPos = (modelViewMatrix*lightPosition).xyz;

    /* normal, tangent and binormal in eye coordinates */

    vec3 N = normalize(normalMatrix*normal.xyz);
    vec3 T = normalize(normalMatrix*objTangent);
    vec3 B = cross(N, T);
```



bumpMap.html (3/5)

```
/* light vector in texture space */

L.x = dot(T, eyeLightPos-eyePosition);
L.y = dot(B, eyeLightPos-eyePosition);
L.z = dot(N, eyeLightPos-eyePosition);

L = normalize(L);

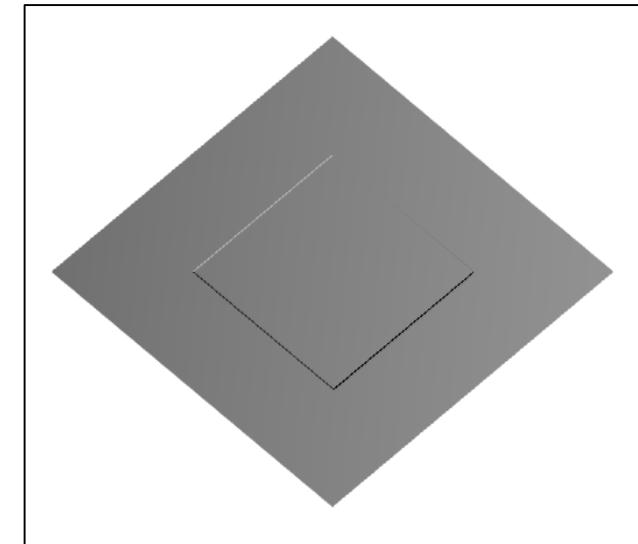
/* view vector in texture space */

V.x = dot(T, -eyePosition);
V.y = dot(B, -eyePosition);
V.z = dot(N, -eyePosition);

V = normalize(V);

}

</script>
```



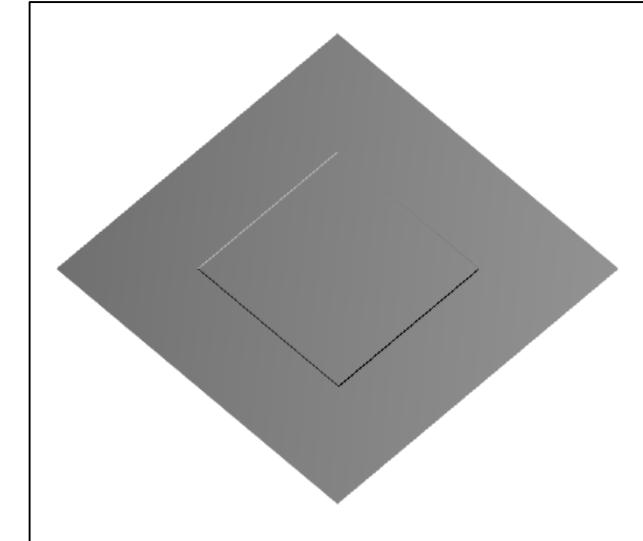
bumpMap.html (4/5)

```
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

varying vec3 L;
varying vec3 V;
varying vec2 fTexCoord;

uniform sampler2D texMap;
uniform vec4 diffuseProduct;

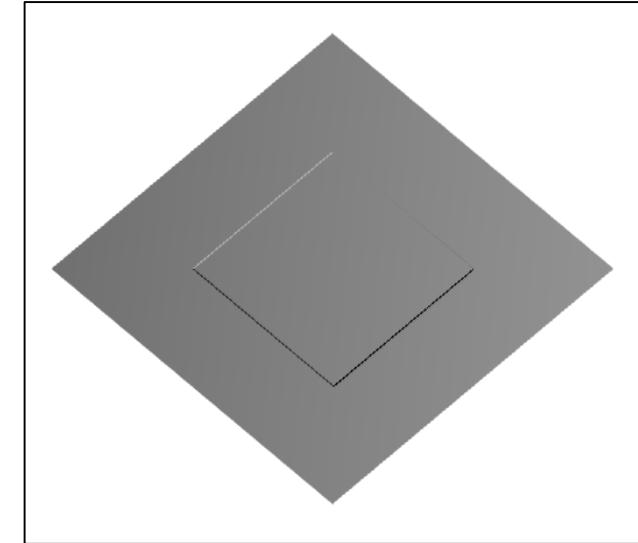
void main()
{
    vec4 N = texture2D(texMap, fTexCoord);
    vec3 NN = normalize(2.0*N.xyz-1.0);
    vec3 LL = normalize(L);
    float Kd = max(dot(NN, LL), 0.0);
    gl_FragColor = vec4(Kd*diffuseProduct.xyz, 1.0);
}
</script>
```



bumpMap.html (5/5)

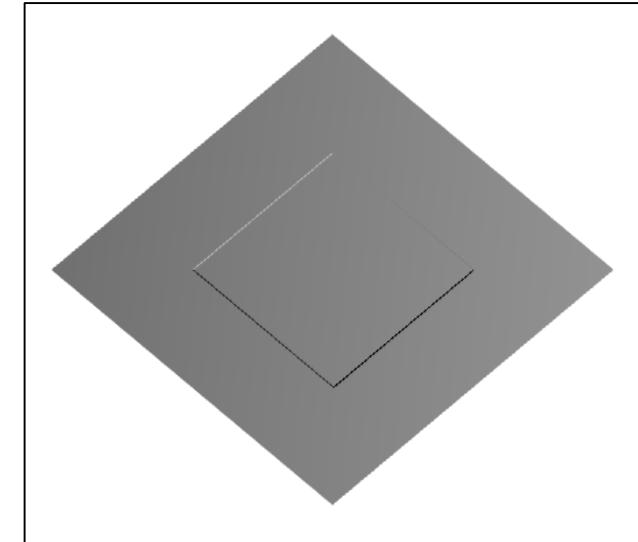
```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/InitShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="bumpMap.js"></script>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



bumpMap.js (1/14)

```
var canvas;  
var gl;  
  
var texSize = 256;  
  
// Bump Data  
  
var data = new Array()  
for (var i = 0; i<= texSize; i++) data[i] = new Array();  
for (var i = 0; i<= texSize; i++)  
    for (var j=0; j<=texSize; j++)  
        data[i][j] = 0.0;  
for (var i = texSize/4; i<3*texSize/4; i++)  
    for (var j = texSize/4; j<3*texSize/4; j++)  
        data[i][j] = 1.0;
```



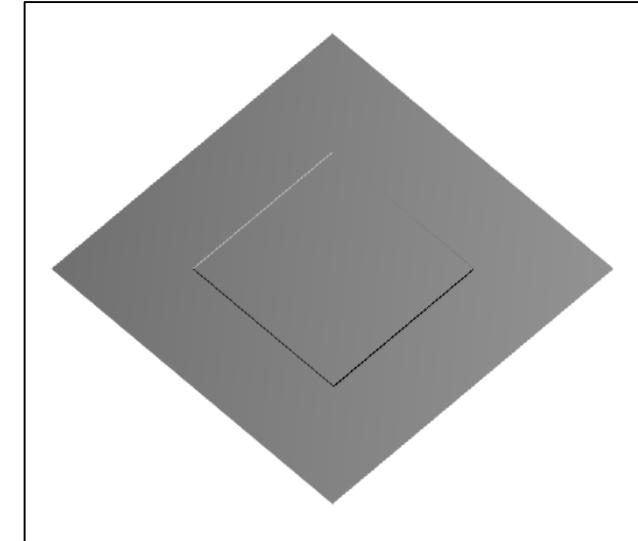
bumpMap.js (2/14)

// Bump Map Normals

```
var normalst = new Array()
for (var i=0; i<texSize; i++) normalst[i] = new Array();
for (var i=0; i<texSize; i++)
  for ( var j = 0; j < texSize; j++)
    normalst[i][j] = new Array();
for (var i=0; i<texSize; i++)
  for ( var j = 0; j < texSize; j++) {
    normalst[i][j][0] = data[i][j]-data[i+1][j];
    normalst[i][j][1] = data[i][j]-data[i][j+1];
    normalst[i][j][2] = 1;
  }
```

// Scale to Texture Coordinates

```
for (var i=0; i<texSize; i++)
  for (var j=0; j<texSize; j++) {
    var d = 0;
    for(k=0;k<3;k++) d+=normalst[i][j][k]*normalst[i][j][k];
    d = Math.sqrt(d);
    for(k=0;k<3;k++) normalst[i][j][k]= 0.5*normalst[i][j][k]/d + 0.5;
  }
```



bumpMap.js (3/14)

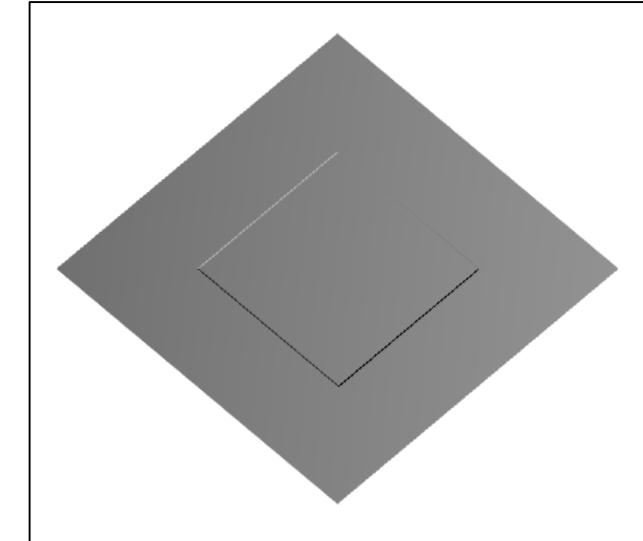
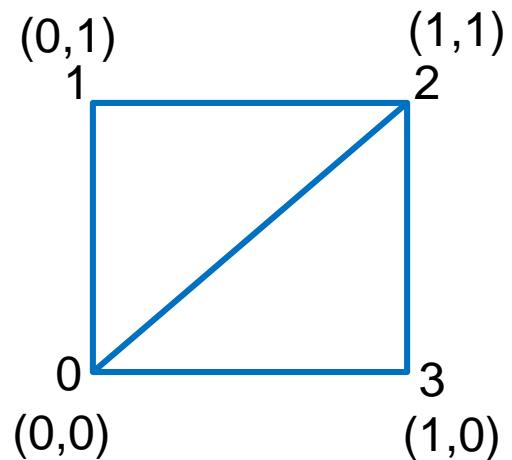
```
// Normal Texture Array
```

```
var normals = new Uint8Array(3*texSize*texSize);
for ( var i = 0; i < texSize; i++ )
    for ( var j = 0; j < texSize; j++ )
        for(var k =0; k<3; k++)
            normals[3*texSize*i+3*j+k] = 255*normalst[i][j][k];
```

```
var numVertices = 6;
```

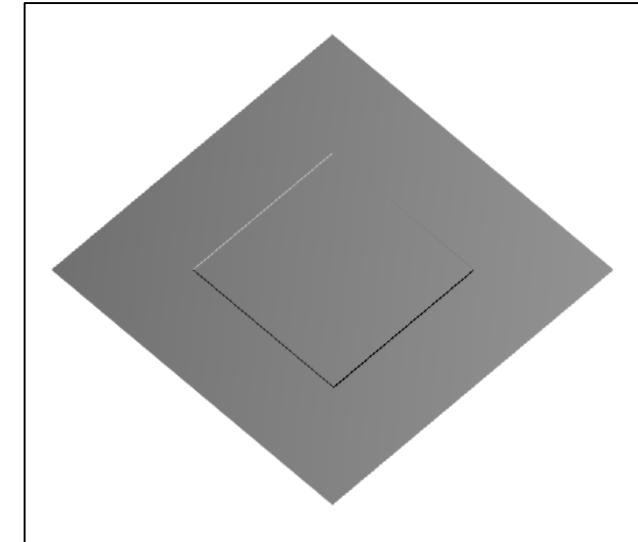
```
var pointsArray = [];
var texCoordsArray = [];
```

```
var texCoord = [
    vec2(0, 0),
    vec2(0, 1),
    vec2(1, 1),
    vec2(1, 0)
];
```



bumpMap.js (4/14)

```
var vertices = [  
    vec4( 0.0, 0.0, 0.0, 1.0 ),  
    vec4( 1.0, 0.0, 0.0, 1.0 ),  
    vec4( 1.0, 0.0, 1.0, 1.0 ),  
    vec4( 0.0, 0.0, 1.0, 1.0 )  
];  
  
var modelViewMatrix, projectionMatrix, normalMatrix;  
  
var eye = vec3(2.0, 2.0, 2.0);  
var at   = vec3(0.5, 0.0, 0.5);  
var up   = vec3(0.0, 1.0, 0.0);
```



bumpMap.js (5/14)

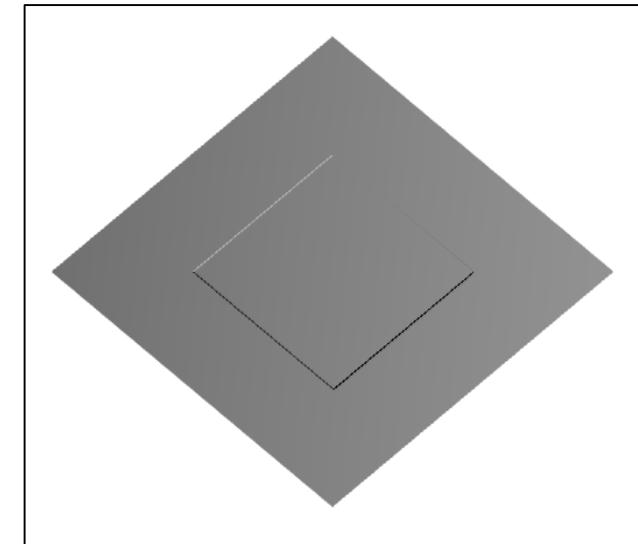
```
var normal = vec4(0.0, 1.0, 0.0, 0.0);
var tangent = vec3(1.0, 0.0, 0.0);

var lightPosition = vec4(0.0, 2.0, 0.0, 1.0 );
var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0 );

var materialDiffuse = vec4( 0.7, 0.7, 0.7, 1.0 );

var program;

var time = 0;
```



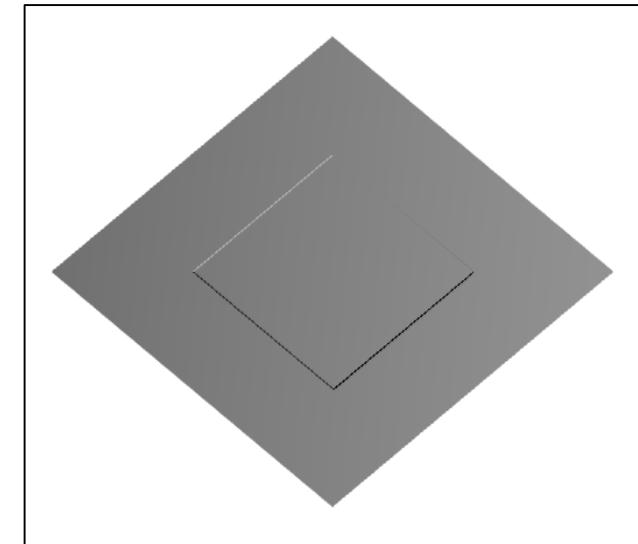
bumpMap.js (6/14)

```
// from glMatrix.js
// Put in MV.js

function mat4ToInverseMat3(mat) {
    dest = mat3();
    var a00 = mat[0][0], a01 = mat[0][1], a02 = mat[0][2];
    var a10 = mat[1][0], a11 = mat[1][1], a12 = mat[1][2];
    var a20 = mat[2][0], a21 = mat[2][1], a22 = mat[2][2];

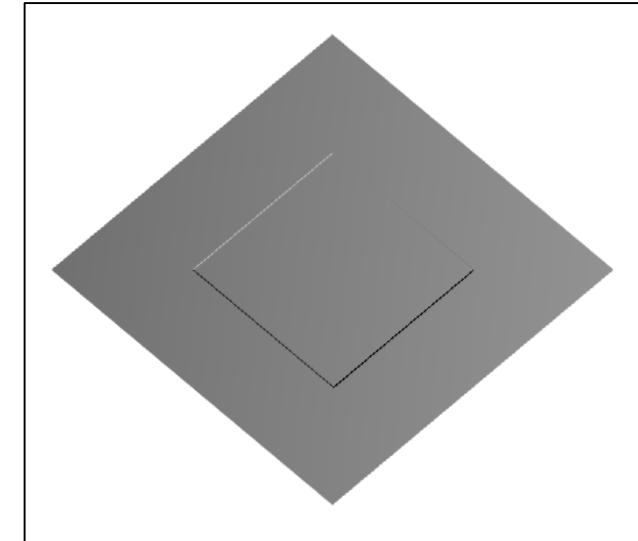
    var b01 = a22*a11-a12*a21;
    var b11 = -a22*a10+a12*a20;
    var b21 = a21*a10-a11*a20;

    var d = a00*b01 + a01*b11 + a02*b21;
    if (!d) { return null; }
    var id = 1/d;
```



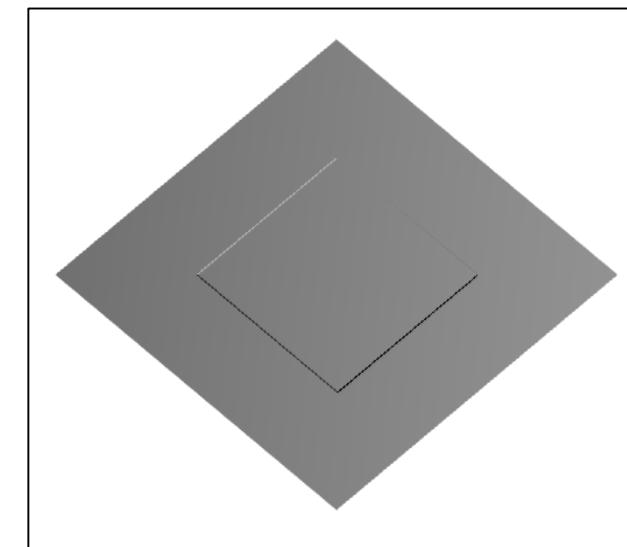
bumpMap.js (7/14)

```
dest[0][0] = b01*id;  
dest[0][1] = (-a22*a01 + a02*a21)*id;  
dest[0][2] = (a12*a01 - a02*a11)*id;  
dest[1][0] = b11*id;  
dest[1][1] = (a22*a00 - a02*a20)*id;  
dest[1][2] = (-a12*a00 + a02*a10)*id;  
dest[2][0] = b21*id;  
dest[2][1] = (-a21*a00 + a01*a20)*id;  
dest[2][2] = (a11*a00 - a01*a10)*id;  
  
return dest;  
}; // end of mat4ToInverseMat3(mat)
```



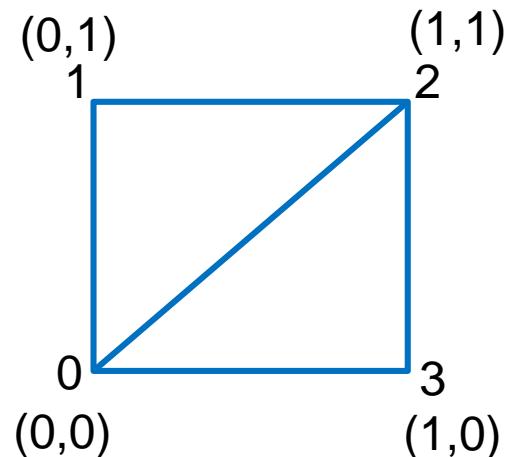
bumpMap.js (8/14)

```
function configureTexture( image ) {
    texture = gl.createTexture();
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, texSize, texSize, 0, gl.RGB, gl.UNSIGNED_BYTE, image);
    gl.generateMipmap(gl.TEXTURE_2D);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
}
```

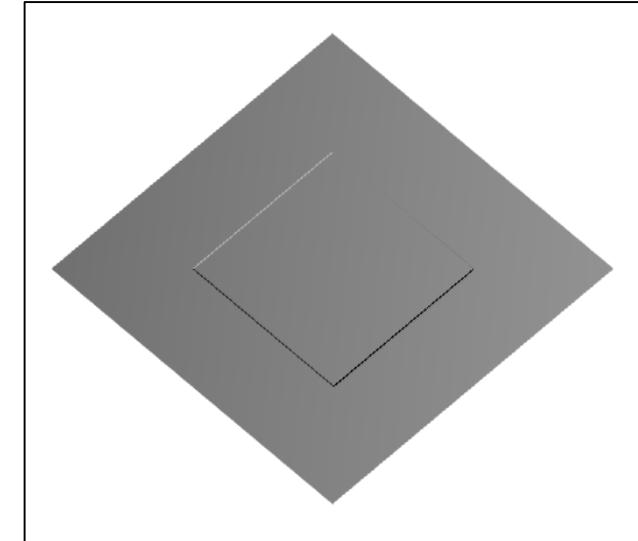


bumpMap.js (9/14)

```
function mesh() {  
    pointsArray.push(vertices[0]);  
    texCoordsArray.push(texCoord[0]);  
  
    pointsArray.push(vertices[1]);  
    texCoordsArray.push(texCoord[1]);  
  
    pointsArray.push(vertices[2]);  
    texCoordsArray.push(texCoord[2]);  
  
    pointsArray.push(vertices[2]);  
    texCoordsArray.push(texCoord[2]);  
  
    pointsArray.push(vertices[3]);  
    texCoordsArray.push(texCoord[3]);  
  
    pointsArray.push(vertices[0]);  
    texCoordsArray.push(texCoord[0]);  
}  
}
```

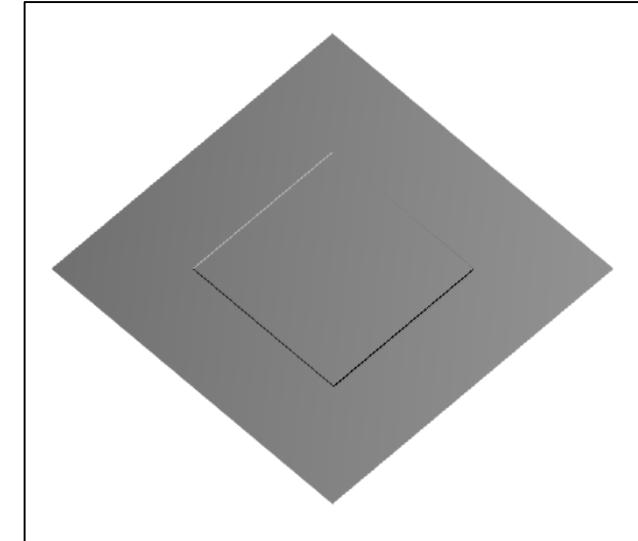


gl.TRIANGLES



bumpMap.js (10/14)

```
window.onload = function init() {  
  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );  
  
    gl.enable(gl.DEPTH_TEST);  
  
    //  
    // Load shaders and initialize attribute buffers  
    //  
    program = initShaders( gl, "vertex-shader", "fragment-shader" );  
    gl.useProgram( program );
```



bumpMap.js (11/14)

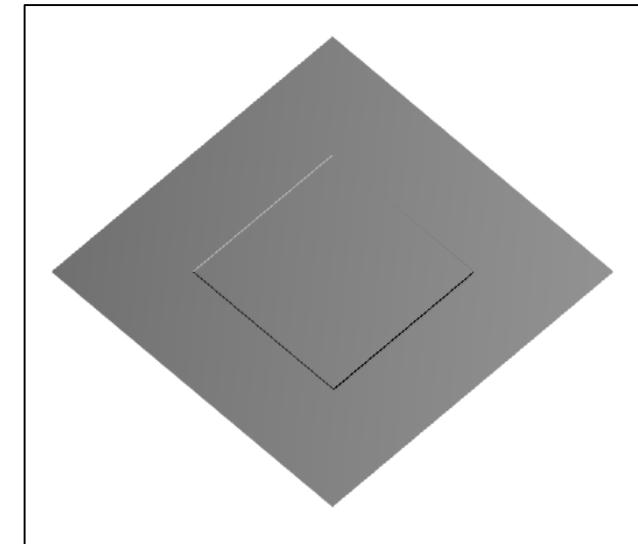
```
modelViewMatrix = lookAt(eye, at, up);
projectionMatrix = ortho(-0.75,0.75,-0.75,0.75,-5.5,5.5);

var normalMatrix = mat4ToInverseMat3(modelViewMatrix);

mesh();

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

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



bumpMap.js (12/14)

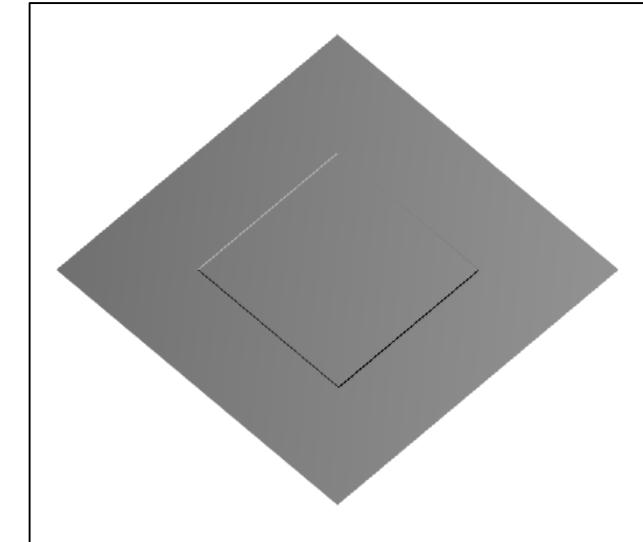
```
var tBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, tBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW);

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

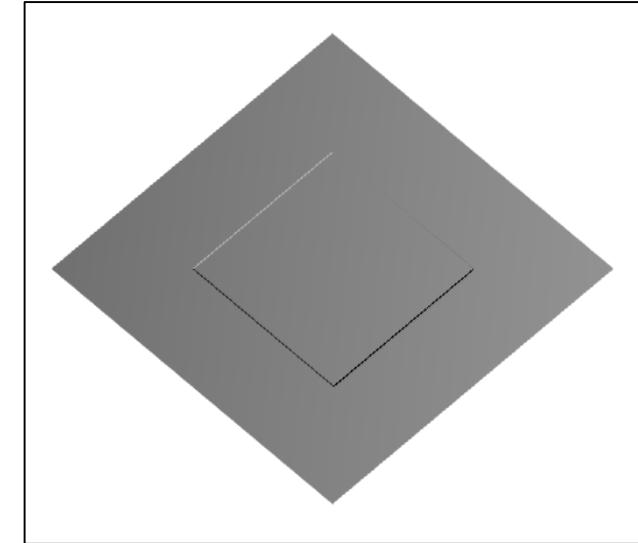
configureTexture(normals);

var diffuseProduct = mult(lightDiffuse, materialDiffuse);

gl.uniform4fv( gl.getUniformLocation(program, "diffuseProduct"), flatten(diffuseProduct));
gl.uniform4fv( gl.getUniformLocation(program, "lightPosition"), flatten(lightPosition));
gl.uniform4fv( gl.getUniformLocation(program, "normal"), flatten(normal));
gl.uniform3fv( gl.getUniformLocation(program, "objTangent"), flatten(tangent));
```



bumpMap.js (13/14)



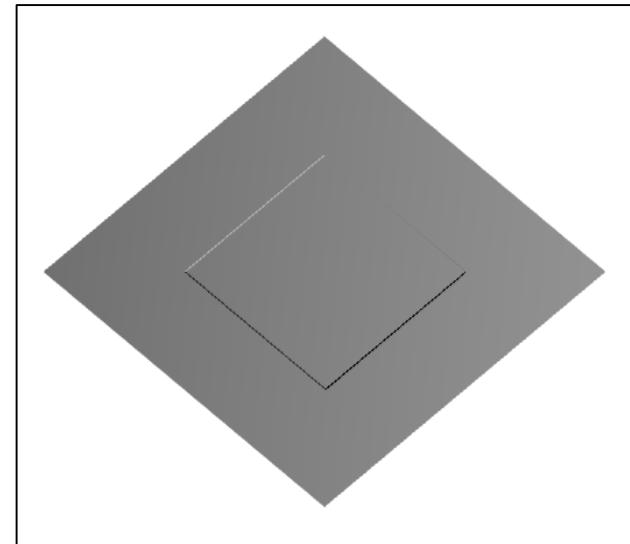
```
gl.uniformMatrix4fv( gl.getUniformLocation(program, "modelViewMatrix"), false, flatten(modelViewMatrix));
gl.uniformMatrix4fv( gl.getUniformLocation(program, "projectionMatrix"), false, flatten(projectionMatrix));
gl.uniformMatrix3fv( gl.getUniformLocation(program, "normalMatrix"), false, flatten(normalMatrix));

render();

} // end of window.onload
```

bumpMap.js (14/14)

```
render = function() {  
  
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
  
    lightPosition[0] = 5.5*Math.sin(0.01*time);  
    lightPosition[2] = 5.5*Math.cos(0.01*time);  
  
    time += 1;  
  
    gl.uniform4fv( gl.getUniformLocation(program, "lightPosition"), flatten(lightPosition) );  
  
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );  
  
    requestAnimFrame(render);  
}  
// end of render()
```



Sample Programs: hatImage1.html, hatImage1.js

sombrero or Mexican hat function $\sin(\pi r)/(\pi r)$

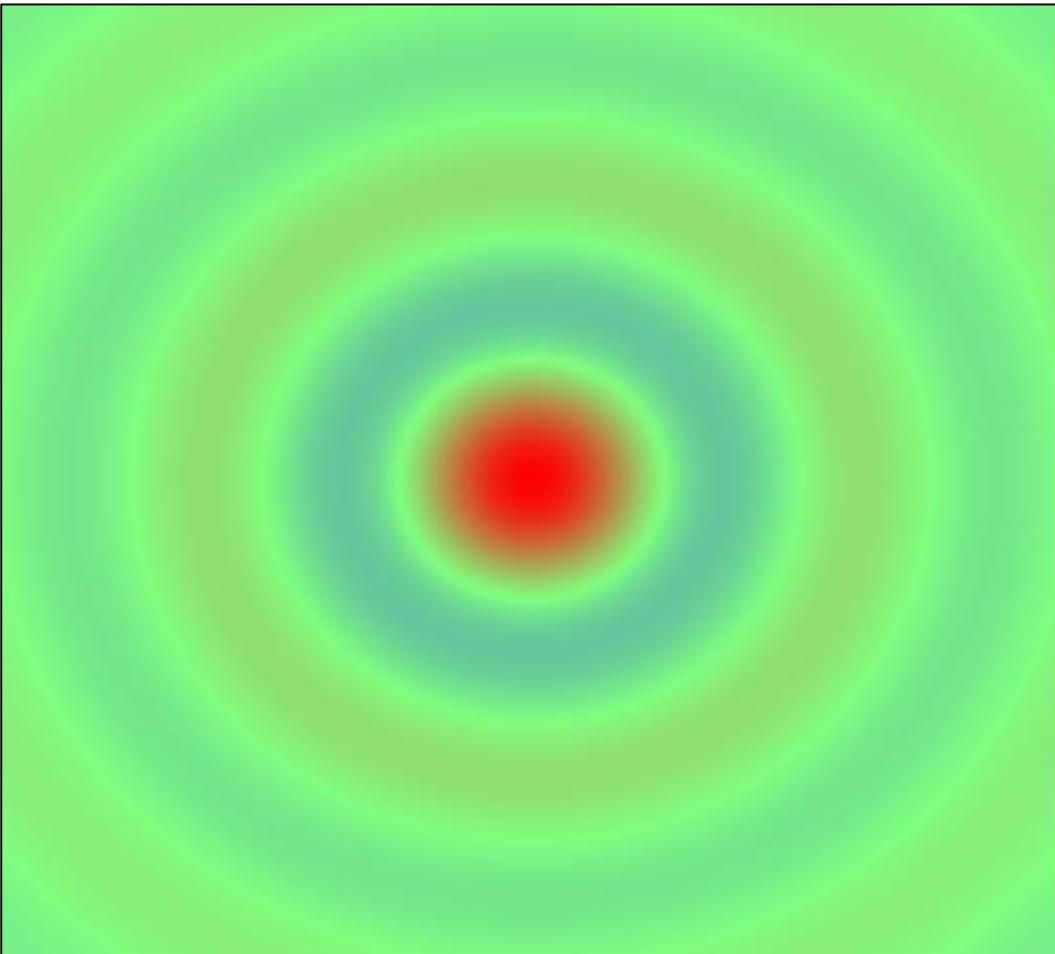


Image of the sombrero function with colors assigned to the y values and the resulting image as texture mapped to a square

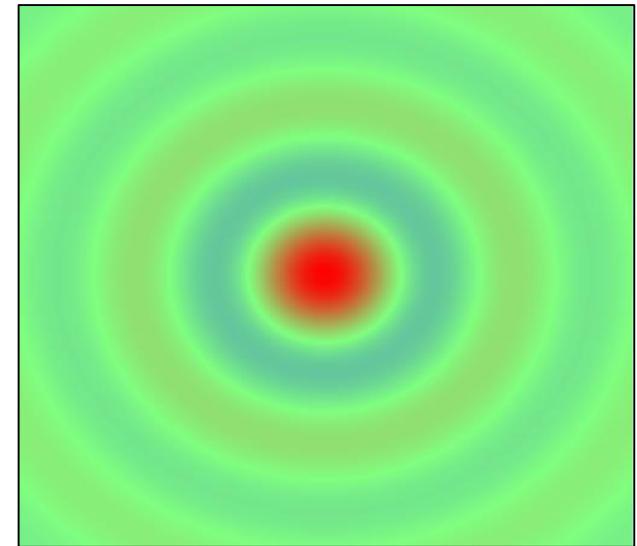
hatImage1.html (1/3)

```
<!DOCTYPE html>
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec2 vPosition;
attribute vec2 vTexCoord;
varying vec2 fTexCoord;

void
main()
{
    fTexCoord = vTexCoord;
    gl_Position = vec4(vPosition.x, vPosition.y, 0.0, 1.0);
}
</script>
```



hatImage1.html (2/3)

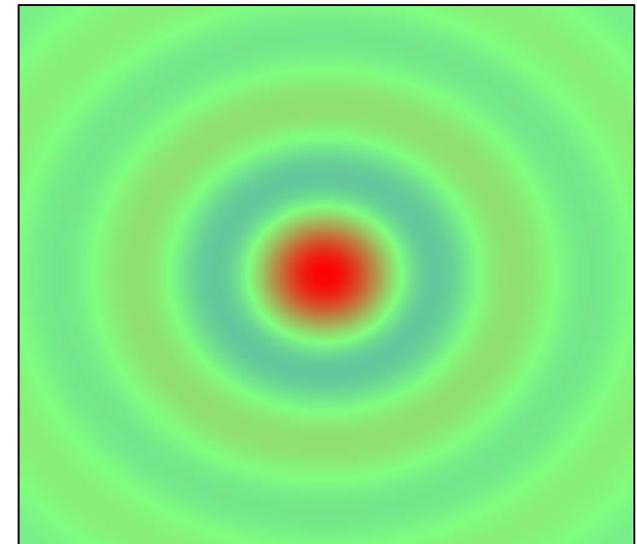
```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

varying vec2 fTexCoord;
uniform sampler2D texture;

void
main()
{
    vec4 color = texture2D(texture, fTexCoord);
    if(color.g<0.5) color.g = 2.0*color.g;
        else color.g = 2.0 - 2.0*color.g;
    color.b = 1.0-color.b;
    gl_FragColor = color;

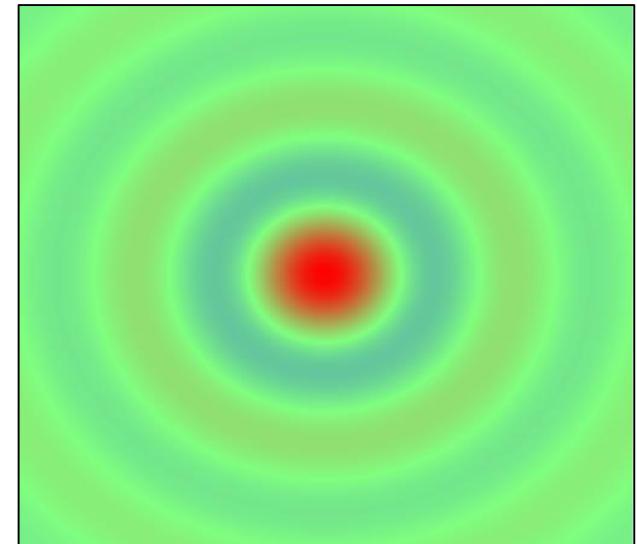
}
</script>
```



hatImage1.html (3/3)

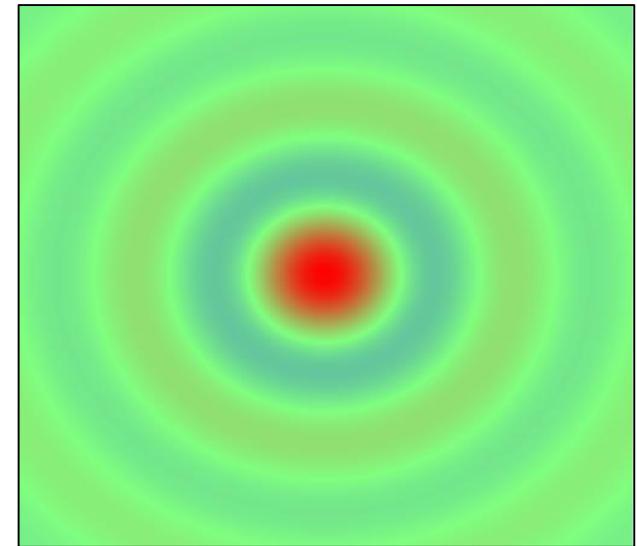
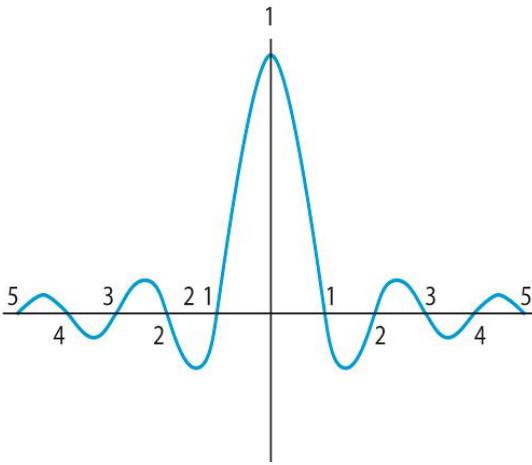
```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="hatImage1.js"></script>

<body>
<canvas id="gl-canvas" width="1024" height="1024">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```

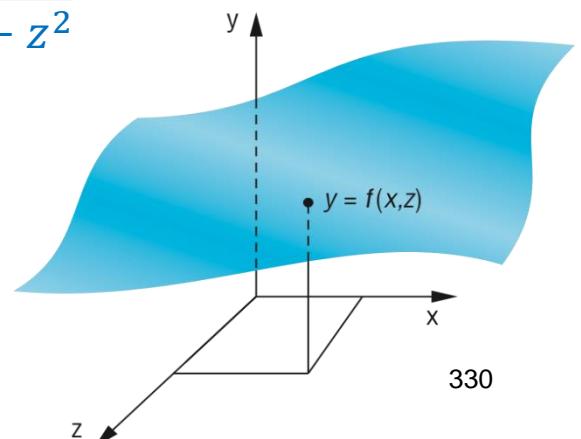
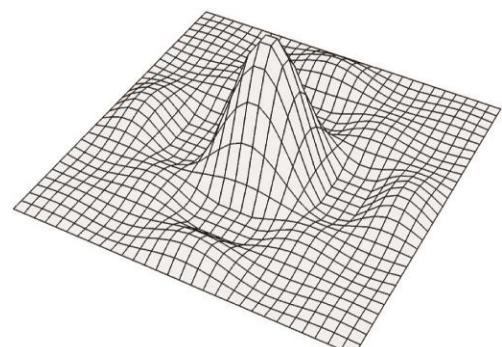


hatImage1.js (1/6)

```
var texSize = 256;  
  
var image = new Uint8Array(texSize*texSize);  
  
for(var i=0; i<texSize; i++) {  
    var x = Math.PI*(8*i/texSize-4.0);  
    for(var j=0; j<texSize; j++) {  
        var y = Math.PI*(8*j/texSize-4.0);  
        var r = Math.sqrt(x*x+y*y);  
        if(r) c = 255*(1+Math.sin( r )/r)/2;  
        else c = 255;  
        image[i*texSize+j] = c;  
    }  
}
```



sombrero or Mexican hat function $f(r) = \sin(\pi r)/(\pi r)$
where $r = \sqrt{x^2 + z^2}$



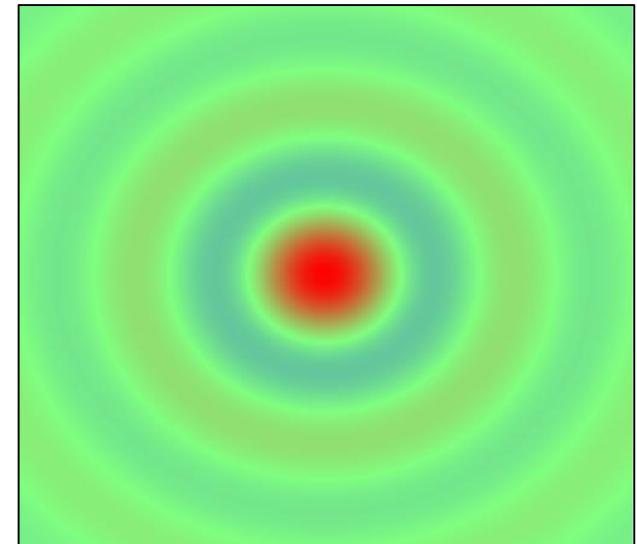
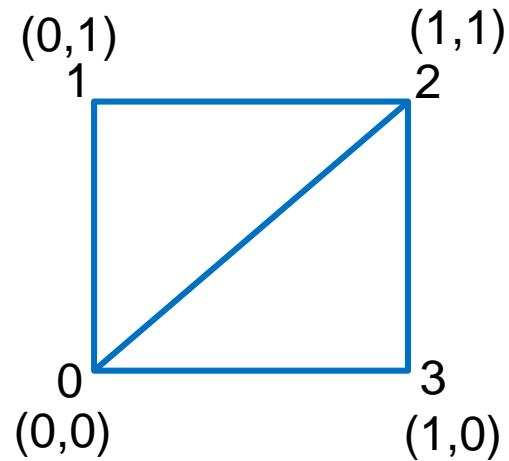
hatImage1.js (2/6)

```
var texCoord = [  
    vec2(0, 0),  
    vec2(0, 1),  
    vec2(1, 1),  
    vec2(1, 0)  
];
```

```
var canvas;  
var gl;
```

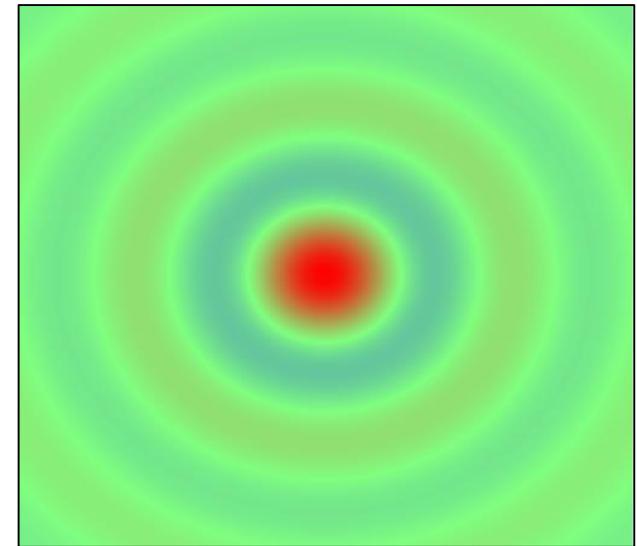
```
var program;
```

```
var pointsArray = [];  
var texCoordsArray = [];
```



hatImage1.js (3/6)

```
window.onload = function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );  
    //  
    // Load shaders and initialize attribute buffers  
    //  
    program = initShaders( gl, "vertex-shader", "fragment-shader" );  
    gl.useProgram( program );
```



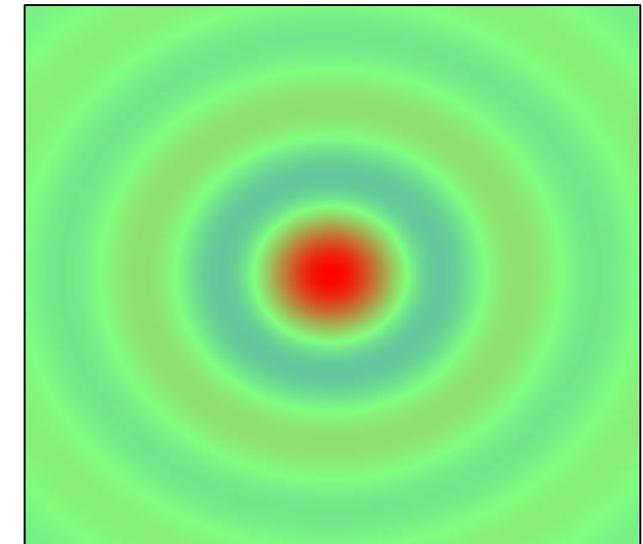
hatImage1.js (4/6)

```
pointsArray.push(vec2(-1, -1));
pointsArray.push(vec2(-1, 1));
pointsArray.push(vec2(1, 1));
pointsArray.push(vec2(1, -1));

texCoordsArray.push(vec2(0, 0));
texCoordsArray.push(vec2(0, 1));
texCoordsArray.push(vec2(1, 1));
texCoordsArray.push(vec2(1, 0));

var tBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, tBuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW);

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



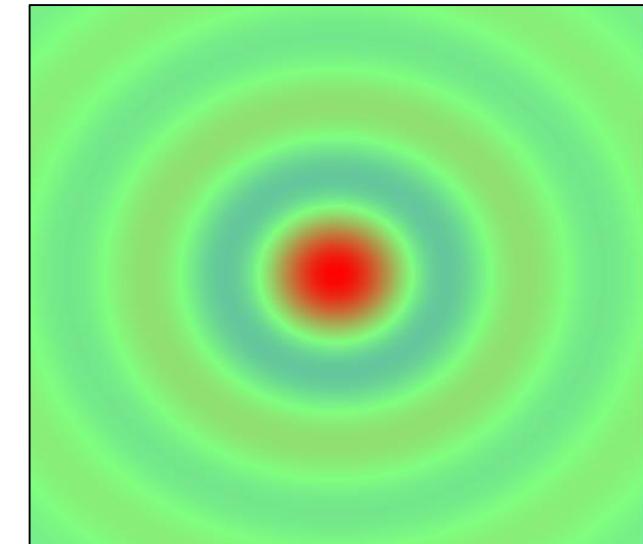
hatImage1.js (5/6)

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

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

var texture = gl.createTexture();
gl.bindTexture( gl.TEXTURE_2D, texture);
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.LUMINANCE, texSize, texSize, 0, gl.LUMINANCE,
              gl.UNSIGNED_BYTE, image);
gl.generateMipmap( gl.TEXTURE_2D );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );

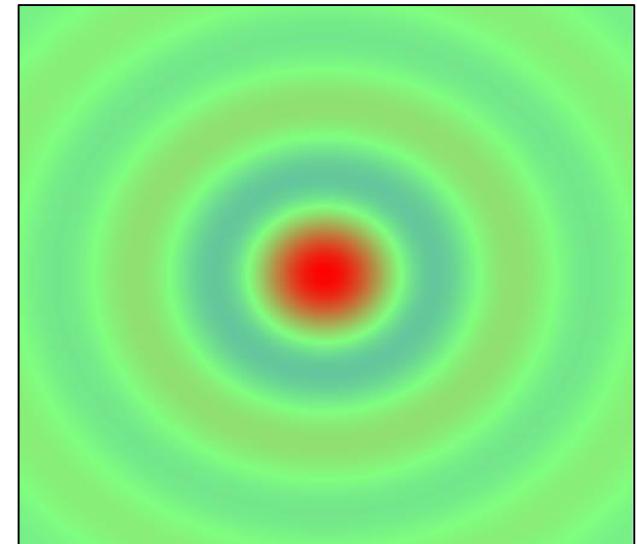
render();
} // end of window.onload
```



A luminance image: each texel has equal red, green, and blue values

hatImage1.js (6/6)

```
function render() {  
  
    gl.clear( gl.COLOR_BUFFER_BIT );  
    gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );  
  
} // end of render()
```



Sample Programs: hatImage2.html, hatImage2.js

sombrero or Mexican hat function $\sin(\pi r)/(\pi r)$

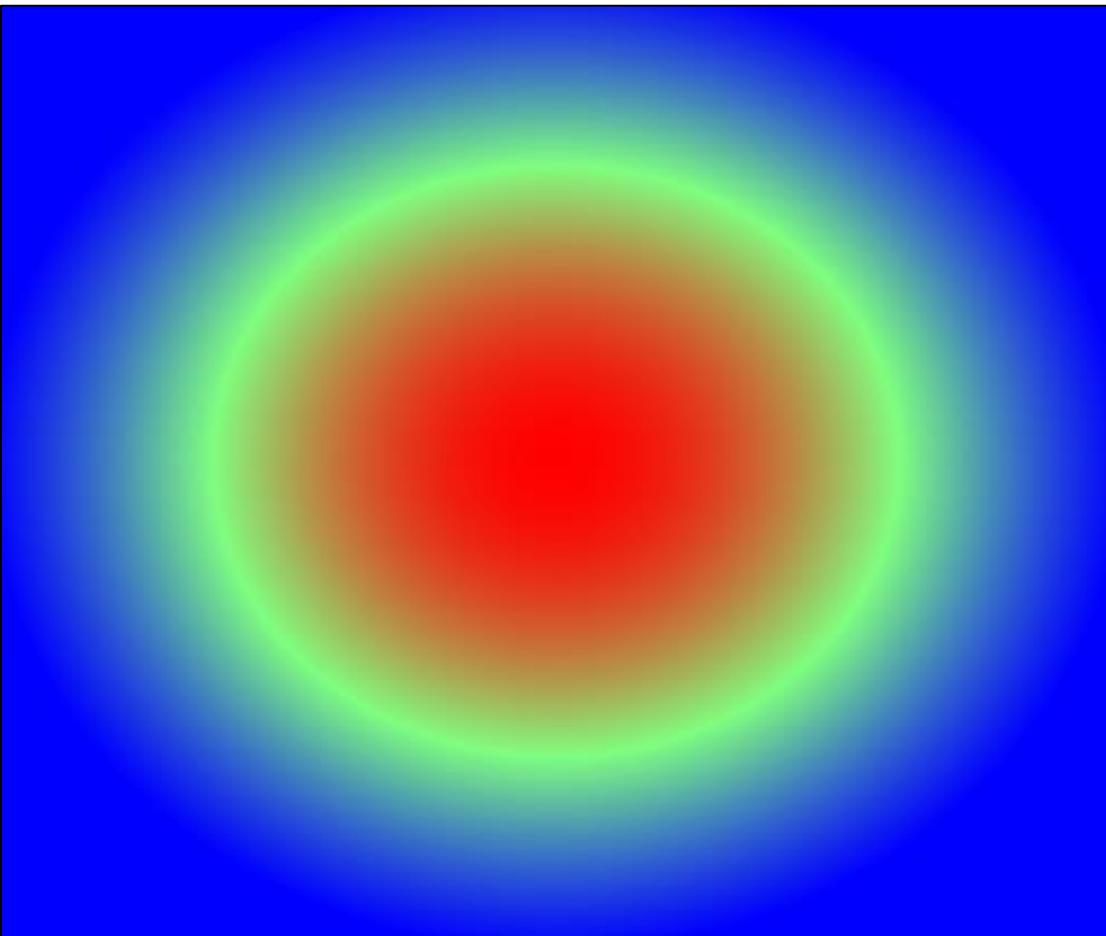


Image of the sombrero function with colors assigned to the y values and the resulting image as texture mapped to a square

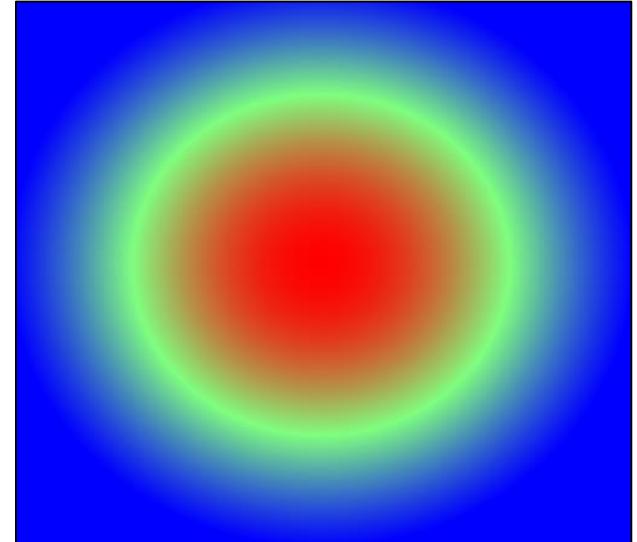
hatImage2.html (1/3)

```
<!DOCTYPE html>
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec2 vPosition;

void
main()
{
    gl_Position = vec4(vPosition.x, vPosition.y, 0.0, 1.0);
}
</script>
```

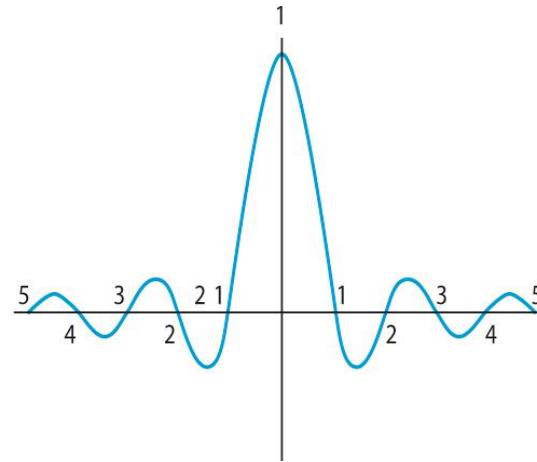


hatImage2.html (2/3)

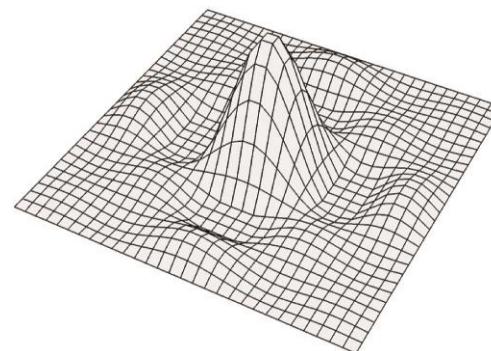
```
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

void main()
{
    float sinc;
    float x = 2.0*gl_FragCoord.x/511.0-1.0;
    float y = 2.0*gl_FragCoord.y/511.0-1.0;
    float r = 2.0*3.14*sqrt(x*x + y*y)/2.0;
    if(r>0.001) sinc = sin(r)/r;
    else sinc = 1.0;

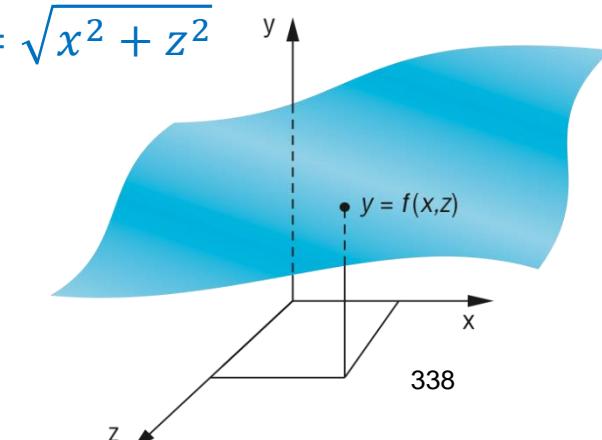
    vec4 color = vec4(sinc, sinc, sinc, 1.0);
    if(color.g<0.5) color.g = 2.0*color.g;
    else color.g = 2.0 - 2.0*color.g;
    color.b = 1.0-color.b;
    gl_FragColor = color;
}
</script>
```



sombrero or Mexican hat function $f(r) = \sin(\pi r)/(\pi r)$



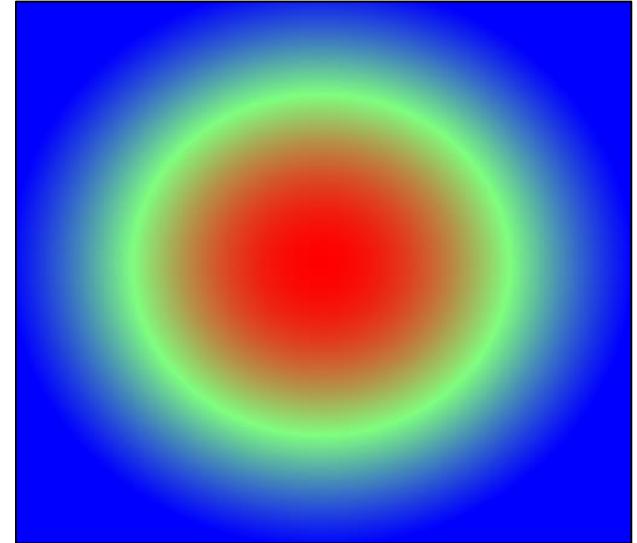
$$\text{where } r = \sqrt{x^2 + z^2}$$



hatImage2.html (3/3)

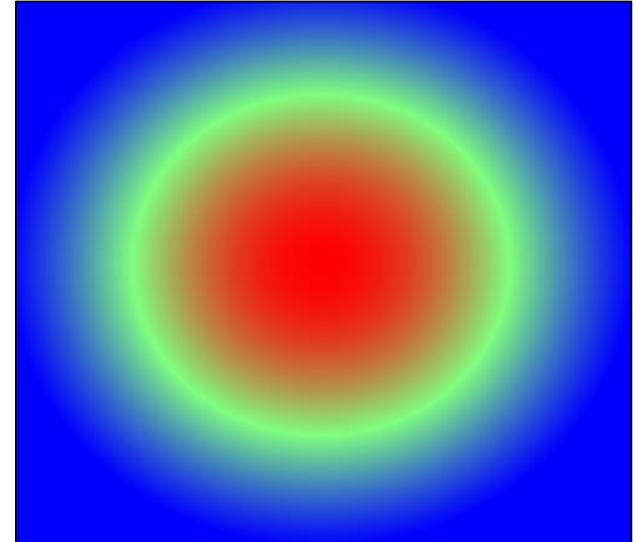
```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="hatImage2.js"></script>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



hatImage2.js (1/3)

```
var canvas;  
var gl;  
  
var program;  
  
var pointsArray = [];  
  
window.onload = function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
```



hatImage2.js (2/3)

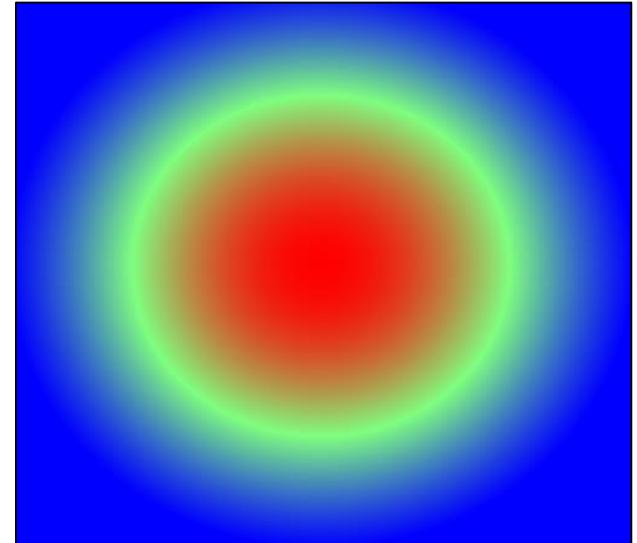
```
// Load shaders and initialize attribute buffers
program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

pointsArray.push(vec2(-1, -1));
pointsArray.push(vec2(-1, 1));
pointsArray.push(vec2(1, 1));
pointsArray.push(vec2(1, -1));

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

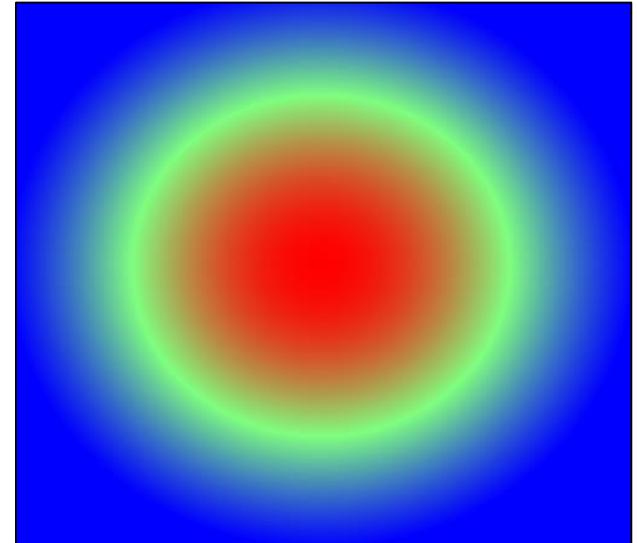
var vPosition = gl.getAttribLocation( program, "vPosition");
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray( vPosition);

render();
} // end of window.onload
```



hatImage2.js (3/3)

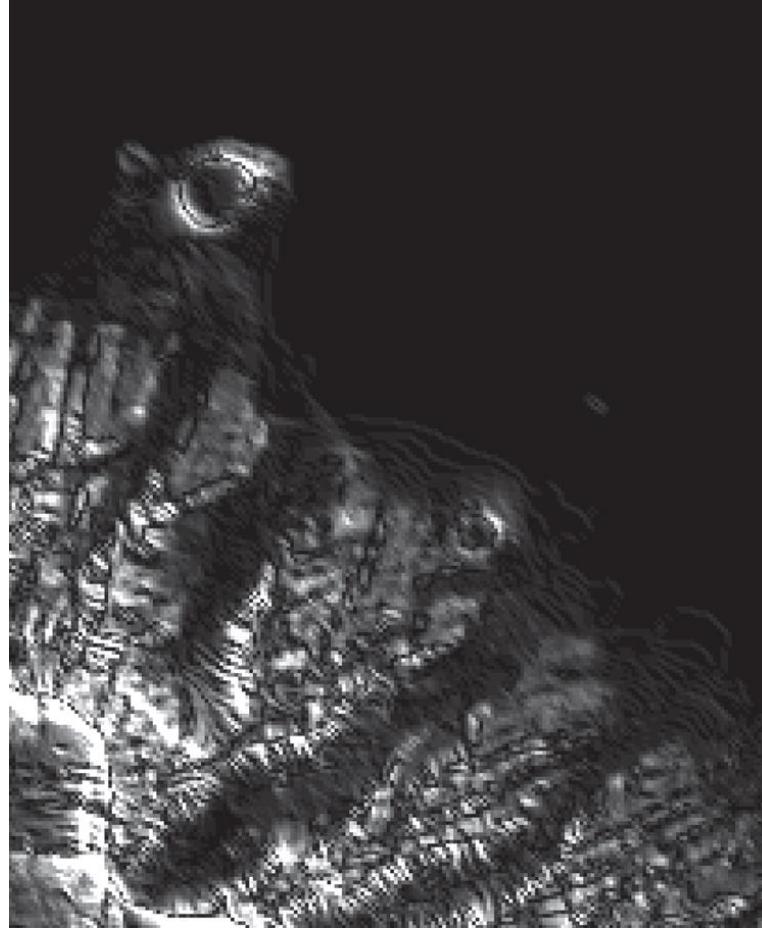
```
function render() {  
  
    gl.clear( gl.COLOR_BUFFER_BIT );  
    gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );  
  
} // end of render()
```



Sample Programs: hawaiimage.html, hawaiimage.js, honolulu4.js



original



enhanced

Image display
of height data
from hawaii
using a texture
map with edge
enhancement
in the fragment
shader

(Honolulu image
after convolution)

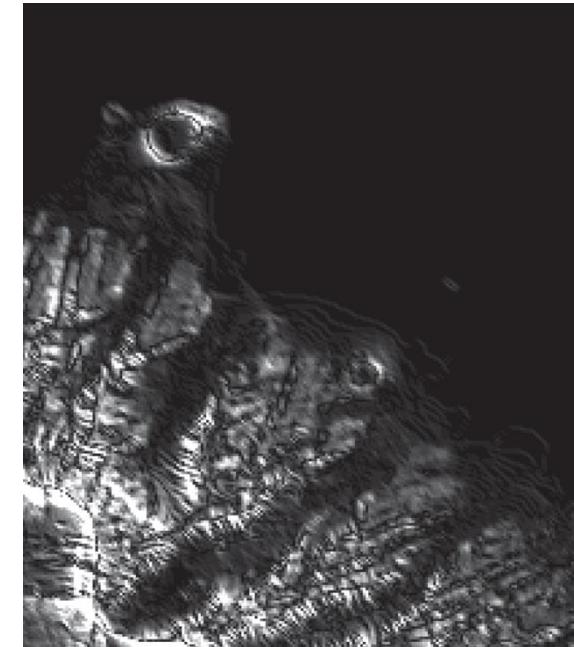
hawaiimage.html (1/3)

```
<!DOCTYPE html>
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec2 vPosition;
attribute vec2 vTexCoord;
varying vec2 fTexCoord;

void
main()
{
    fTexCoord = vTexCoord;
    gl_Position = vec4(vPosition.x, vPosition.y, 0.0, 1.0);
}
</script>
```



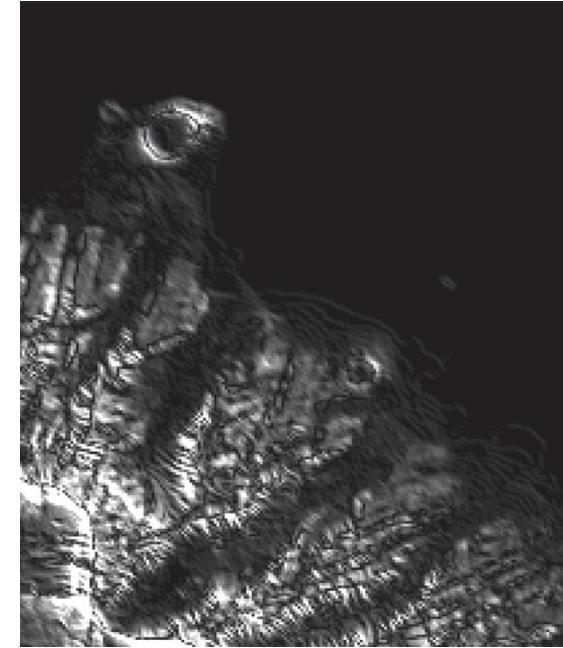
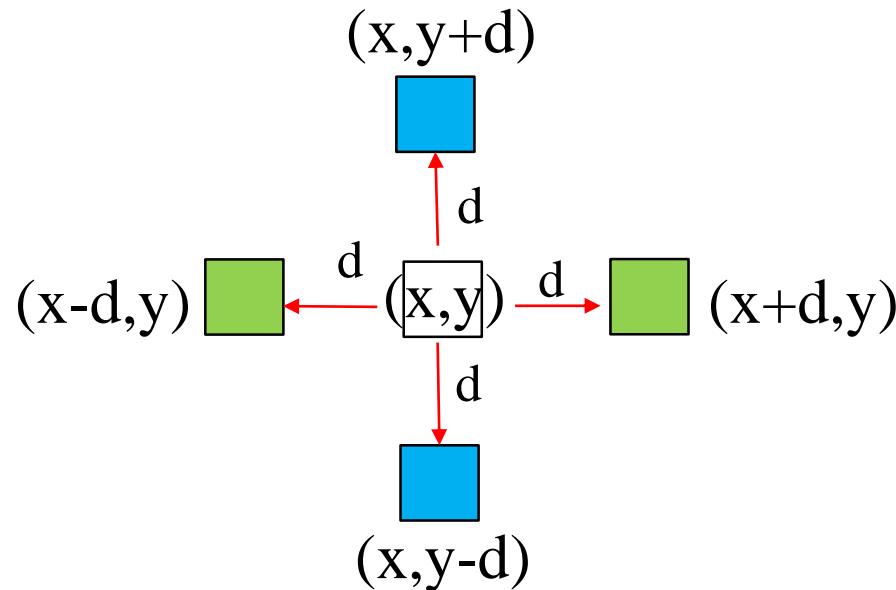
hawaiimage.html (2/3)

```
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

varying vec2 fTexCoord;
uniform sampler2D texture;

void main()
{
    float d = 1.0/256.0;
    float x = fTexCoord.x;
    float y = fTexCoord.y;

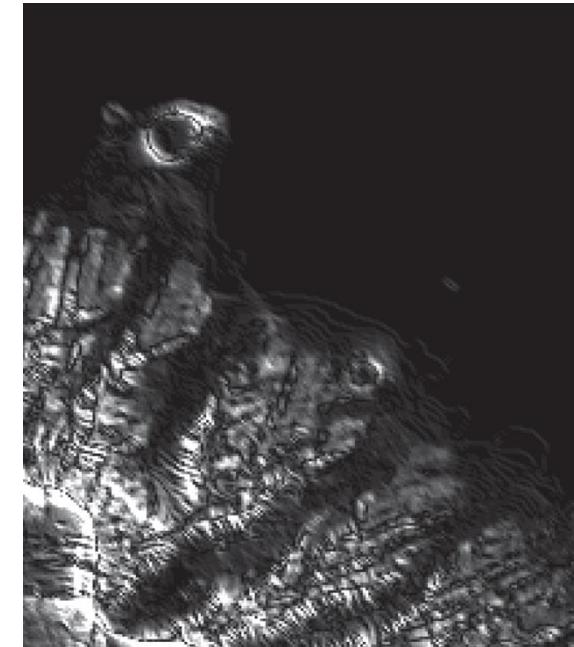
    gl_FragColor = 10.0*abs( texture2D( texture, vec2(x+d, y))-texture2D( texture, vec2(x-d, y)))
                    +10.0*abs( texture2D( texture, vec2(x, y+d))-texture2D( texture, vec2(x, y-d)));
    gl_FragColor.w = 1.0;
}
</script>
```



hawaiimage.html (3/3)

```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="honolulu4.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="hawaiimage.js"></script>
```

```
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



honolulu4.js (1/1)

```
var nRows = 256;
var nColumns = 256;

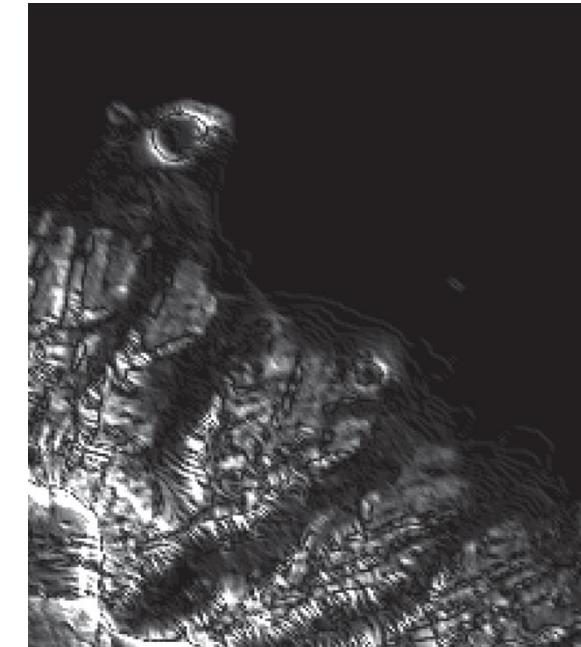
var data = new Uint8Array(nRows*nColumns);

var max = 0;

var rawData = new Uint32Array([-----]);

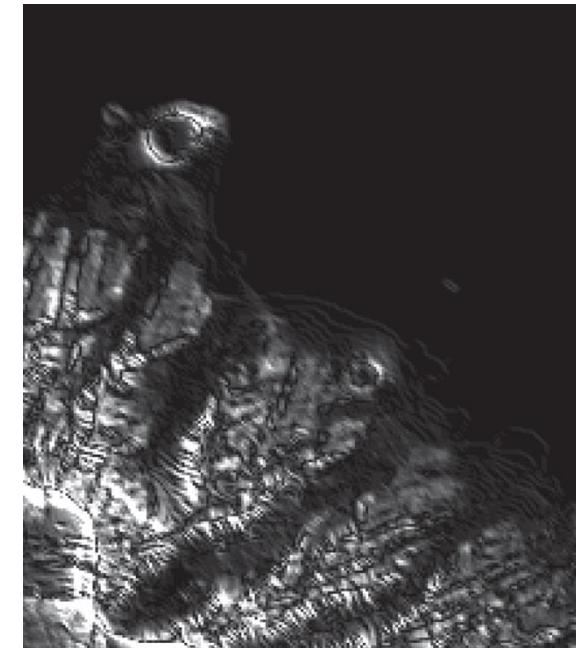
for(var i=0; i<nRows;i++) for(var j=0; j<nColumns; j++) {
    if( rawData[i*nColumns+j]>max) max = rawData[i*nColumns+j];
    //console.log(rawData[i*nColumns+j]);
}
//console.log(max);

for(var i=0; i<nRows;i++) for(var j=0; j<nColumns; j++) {
    data[i*nColumns+j] = 255*(rawData[i*nColumns+j]/max);
    //console.log(data[i*nColumns+j]);
}
```



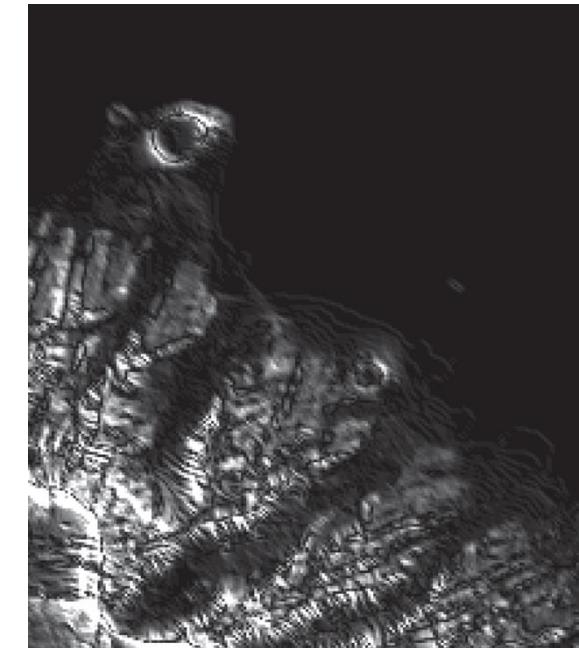
hawaiimage.js (1/5)

```
var texSize = 256;  
  
var canvas;  
var gl;  
  
var program;  
  
var pointsArray = [];  
var texCoordsArray = [];  
  
window.onload = function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL(canvas);  
    if (!gl) {alert( "WebGL isn't available");}  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
```



hawaiimage.js (2/5)

```
//  
// Load shaders and initialize attribute buffers  
//  
program = initShaders( gl, "vertex-shader", "fragment-shader" );  
gl.useProgram( program );  
  
pointsArray.push(vec2(-1, -1));  
pointsArray.push(vec2(-1, 1));  
pointsArray.push(vec2(1, 1));  
pointsArray.push(vec2(1, -1));  
  
texCoordsArray.push(vec2(0, 0));  
texCoordsArray.push(vec2(0, 1));  
texCoordsArray.push(vec2(1, 1));  
texCoordsArray.push(vec2(1, 0));
```



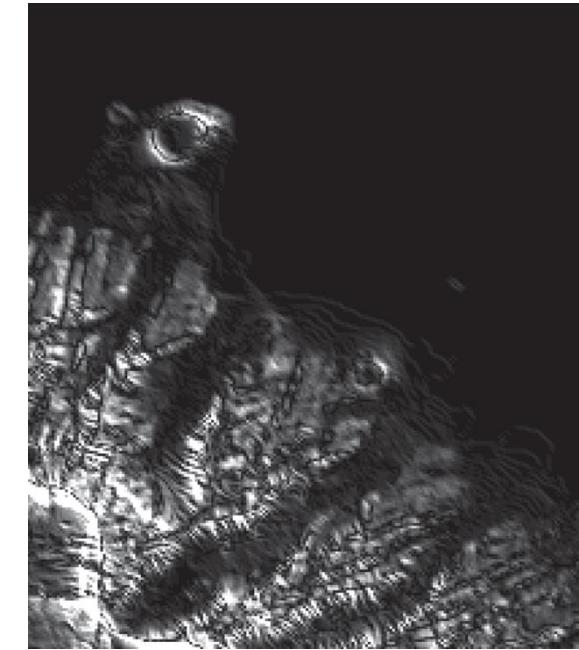
hawaiimage.js (3/5)

```
var tBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, tBuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW);

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

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

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

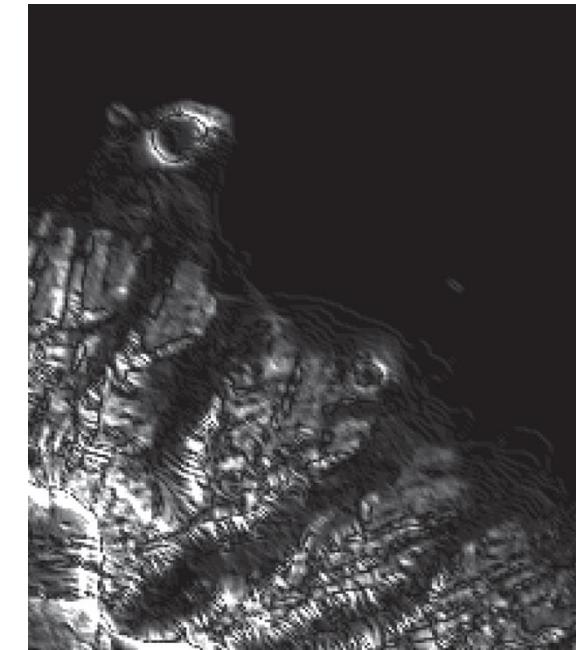


hawaiimage.js (4/5)

```
var textureId = gl.createTexture();
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, textureId);

gl.texImage2D(gl.TEXTURE_2D, 0, gl.LUMINANCE, texSize, texSize, 0, gl.LUMINANCE,
             gl.UNSIGNED_BYTE, data);
gl.generateMipmap( gl.TEXTURE_2D );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );

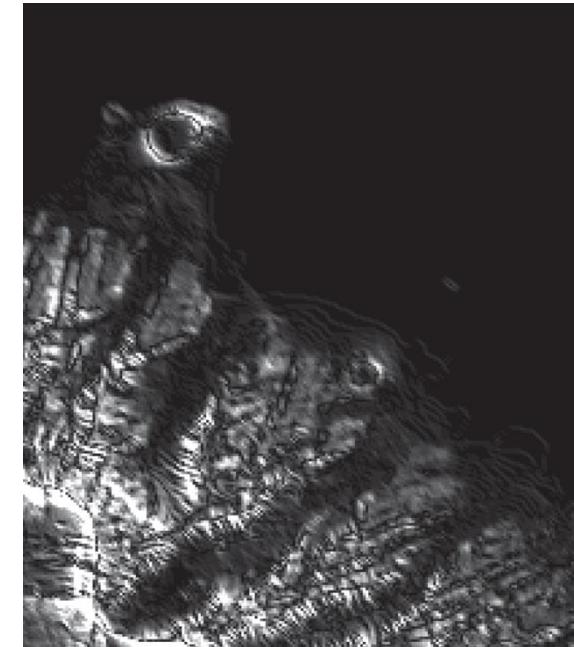
render();
} // end of window.onload
```



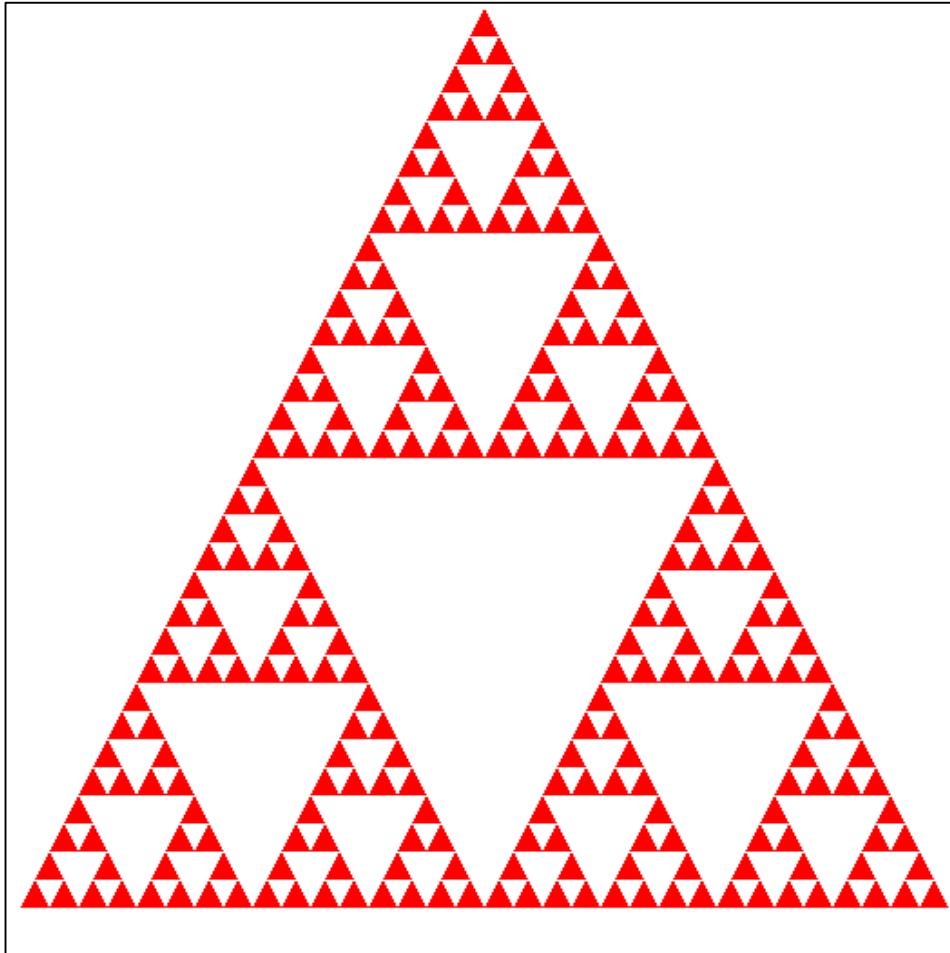
A luminance image: each texel has equal red, green, and blue values

hawaiimage.js (5/5)

```
function render() {  
  
    gl.clear( gl.COLOR_BUFFER_BIT );  
    gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );  
  
} // end of render()
```



Sample Programs: render1.html, render1.js



Sierpinski gasket rendered to a texture and then display on a square

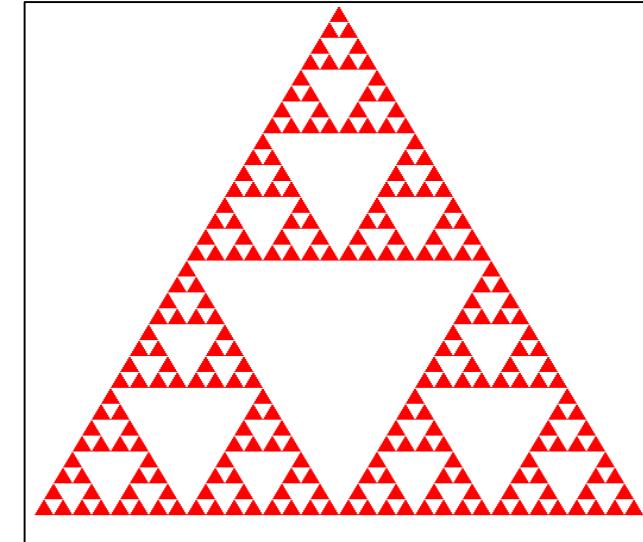
render1.html (1/5)

```
<!DOCTYPE html>
<html>

<script id="vertex-shader1" type="x-shader/x-vertex">

attribute vec4 vPosition;

void main()
{
    gl_Position = vPosition;
}
</script>
```



render1.html (2/5)

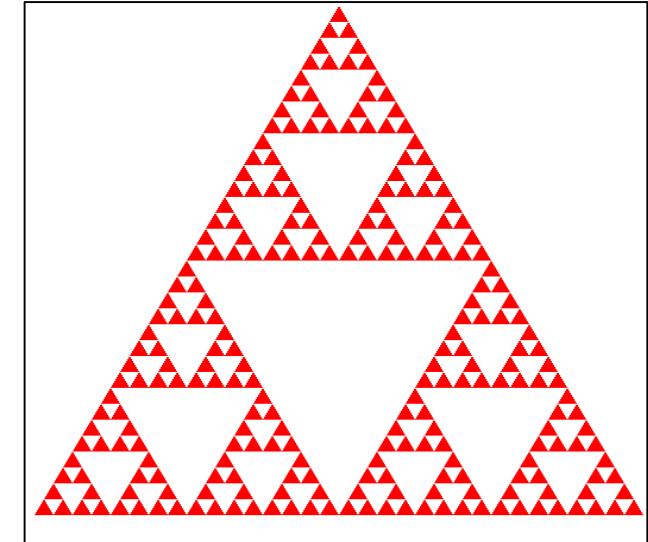
```
<script id="vertex-shader2" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec2 vTexCoord;

varying vec2 fTexCoord;

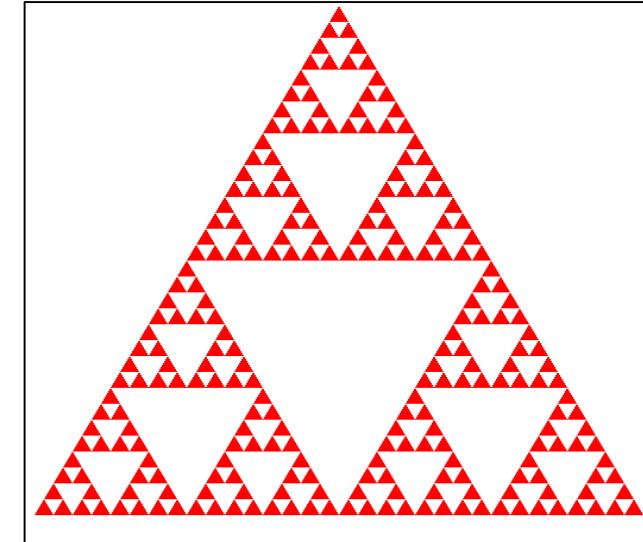
void main()
{
    gl_Position = vPosition;
    fTexCoord = vTexCoord;

}
</script>
```



render1.html (3/5)

```
<script id="fragment-shader1" type="x-shader/x-fragment">  
  
precision mediump float;  
  
void  
main()  
{  
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);  
}  
</script>
```



render1.html (4/5)

```
<script id="fragment-shader2" type="x-shader/x-fragment">

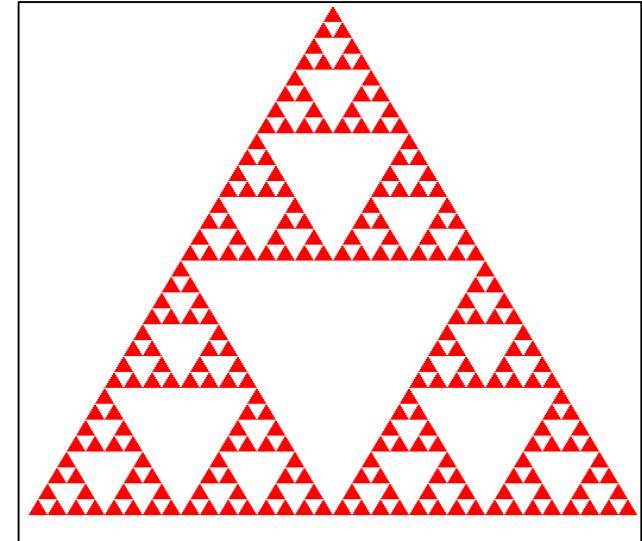
precision mediump float;

varying vec2 fTexCoord;

uniform sampler2D texture;

void main()
{
    gl_FragColor = texture2D( texture, fTexCoord);

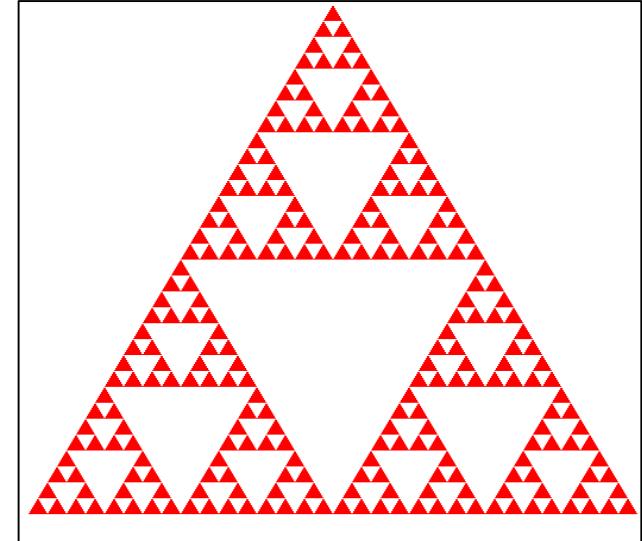
}
</script>
```



render1.html (5/5)

```
<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="render1.js"></script>

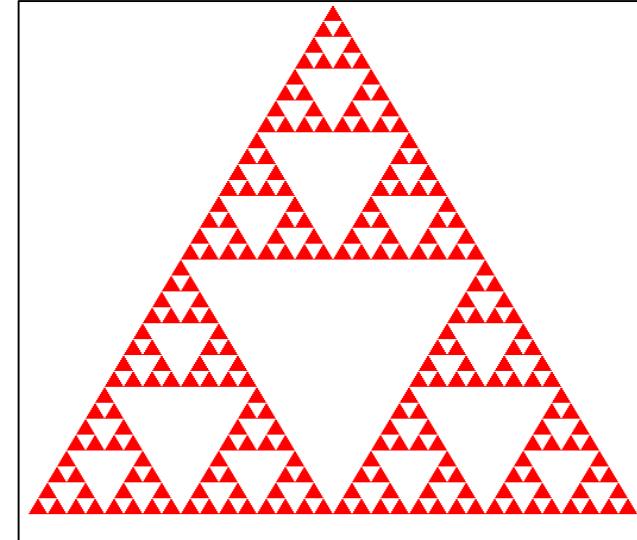
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



render1.js (1/13)

```
// Note that because this example is 2D, the renderbuffer is not needed  
// I have commented out the relevant lines of code  
// If you are doing a 3D app and need a depth buffer then uncomment these lines
```

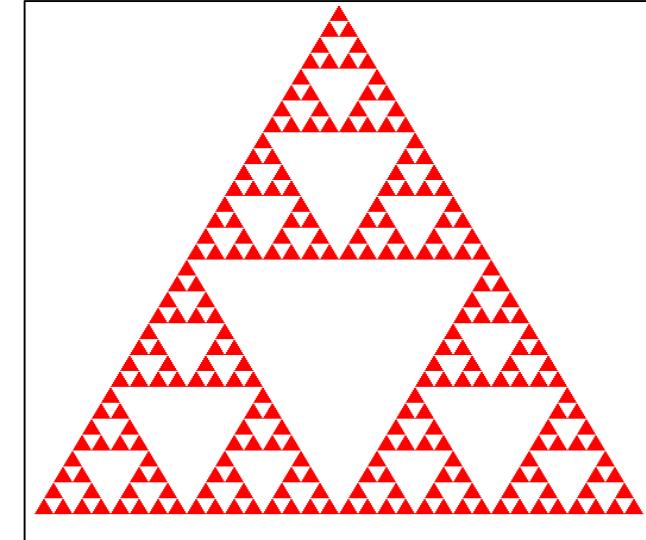
```
var canvas;  
var gl;  
  
var numTimesToSubdivide = 5;  
var numTriangles = 243; //  $3^5$  triangles generated  
var numVertices = 3 * numTriangles;  
  
var Index = 0;
```



render1.js (2/13)

```
var texCoord = [  
    vec2(0, 0),  
    vec2(0, 1),  
    vec2(1, 1),  
    vec2(1, 1),  
    vec2(1, 0),  
    vec2(0, 0)  
];
```

```
var vertices = [  
    vec2( -1, -1 ),  
    vec2( -1, 1 ),  
    vec2( 1, 1 ),  
    vec2( 1, 1 ),  
    vec2( 1, -1 ),  
    vec2( -1, -1 )  
];
```



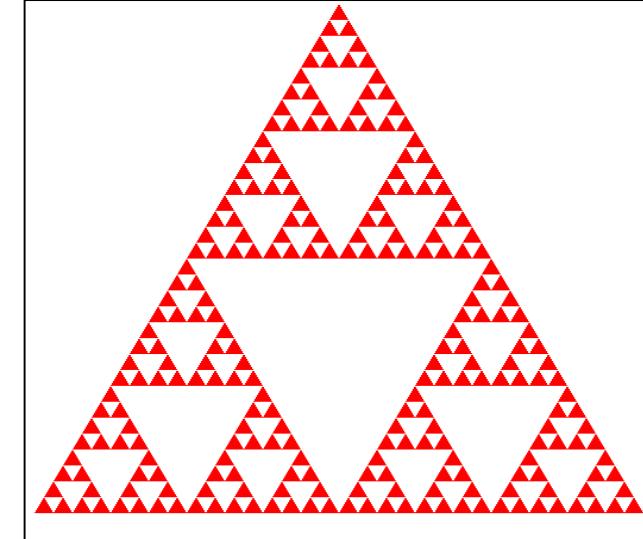
render1.js (3/13)

```
var pointsArray = [];

var program1, program2;
var framebuffer, texture1;
var buffer1, buffer2, buffer3;

//var renderbuffer;

function triangle( a, b, c ) {
    pointsArray.push( a, b, c );
}
```

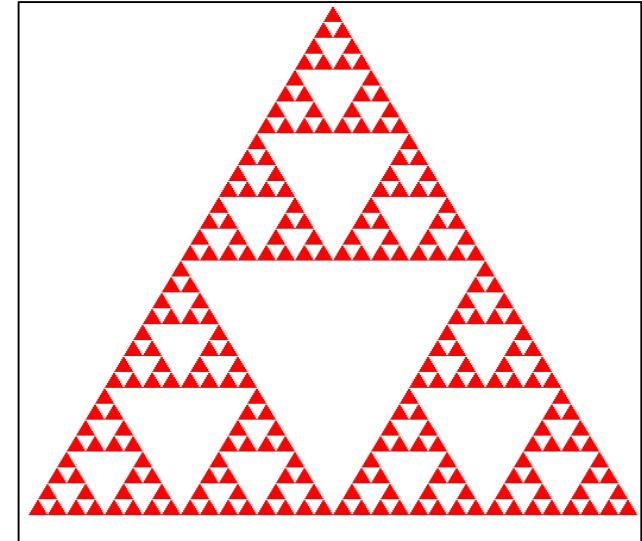


render1.js (4/13)

```
function divideTriangle( a, b, c, count )
{
    if ( count == 0 ) {
        triangle( a, b, c );
    }
    else {
        var ab = mix( a, b, 0.5 );
        var ac = mix( a, c, 0.5 );
        var bc = mix( b, c, 0.5 );

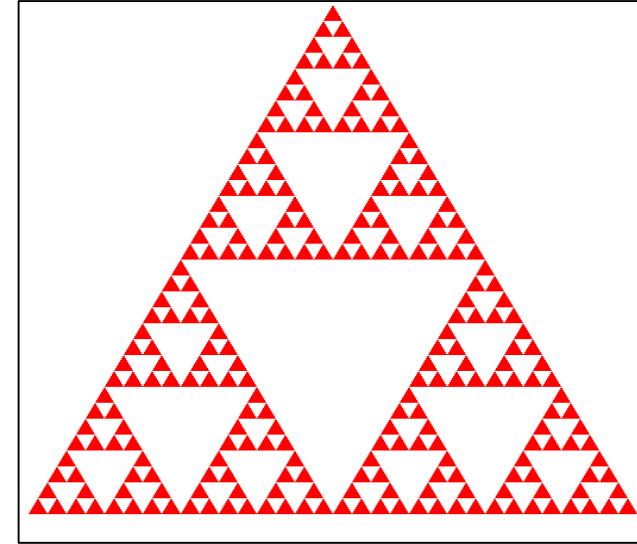
        --count;

        divideTriangle( a, ab, ac, count );
        divideTriangle( c, ac, bc, count );
        divideTriangle( b, bc, ab, count );
    }
}
```



render1.js (5/13)

```
window.onload = function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    // Create an empty texture  
  
    texture1 = gl.createTexture();  
    gl.activeTexture( gl.TEXTURE0 );  
    gl.bindTexture( gl.TEXTURE_2D, texture1 );  
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);  
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 512, 512, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);  
    gl.generateMipmap(gl.TEXTURE_2D);  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );  
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST )
```



render1.js (6/13)

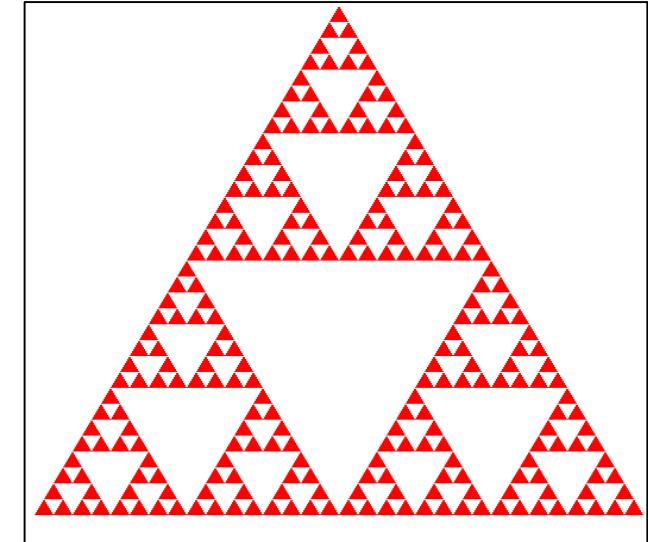
```
// Allocate a frame buffer object
```

```
framebuffer = gl.createFramebuffer();
gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
framebuffer.width = 512;
framebuffer.height = 512;
```

```
//renderbuffer = gl.createRenderbuffer();
//gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);
//gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16, 512, 512);
```

```
// Attach color buffer
```

```
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture1, 0);
//gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.RENDERBUFFER, renderbuffer);
```



render1.js (7/13)

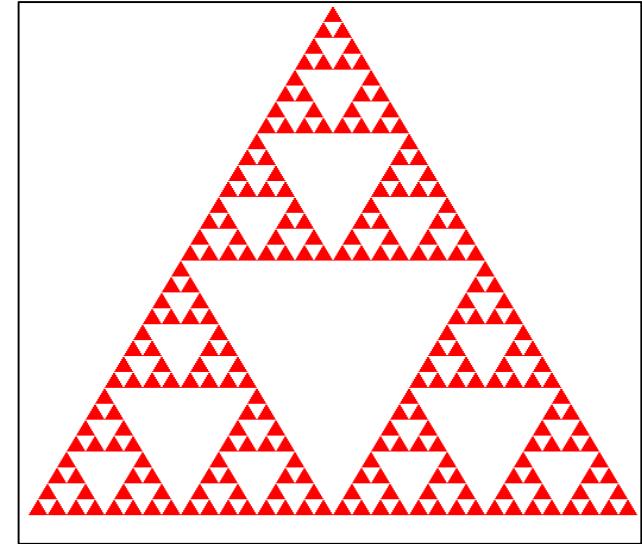
```
// check for completeness

var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);
if(status != gl.FRAMEBUFFER_COMPLETE) alert('Frame Buffer Not Complete');

// Load shaders and initialize attribute buffers
program1 = initShaders( gl, "vertex-shader1", "fragment-shader1" );
program2 = initShaders( gl, "vertex-shader2", "fragment-shader2" );

gl.useProgram( program1 );

var vertices2 = [
    vec2(-1, -1),
    vec2( 0, 1),
    vec2( 1, -1)
];
divideTriangle(vertices2[0], vertices2[1], vertices2[2], numTimesToSubdivide);
```



render1.js (8/13)

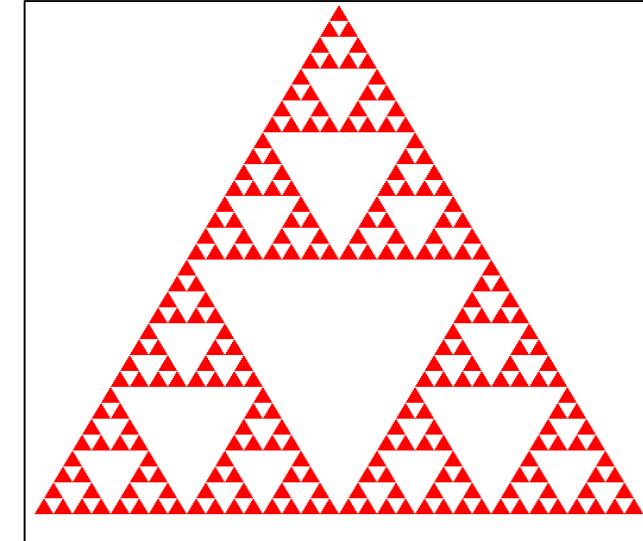
```
// Create and initialize a buffer object
```

```
buffer1 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer1 );
gl.bufferData( gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW );
```

```
var vPosition = gl.getAttribLocation( program1, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
```

```
// Bind FBO and render
```

```
gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
//gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);
```



render1.js (9/13)

```
gl.viewport(0, 0, 512, 512);
gl.clearColor(1.0, 1.0, 1.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT );

gl.drawArrays(gl.TRIANGLES, 0, numVertices);

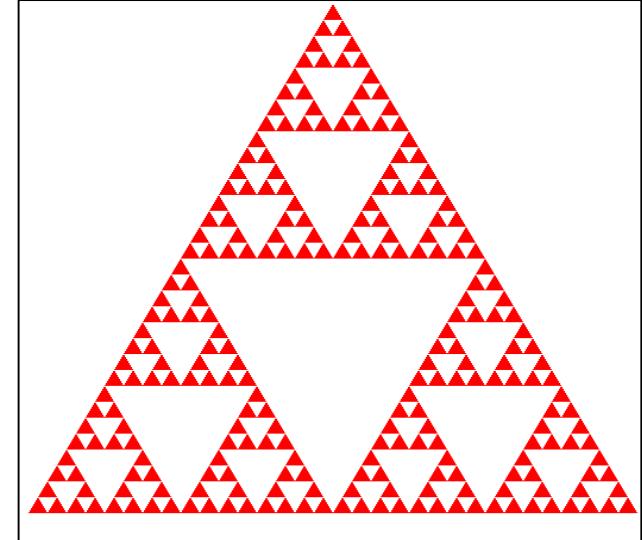
// Bind to window system frame buffer, unbind the texture

gl.bindFramebuffer(gl.FRAMEBUFFER, null);

//gl.bindRenderbuffer(gl.RENDERBUFFER, null);

//gl.deleteFramebuffer(framebuffer);
//gl.deleteRenderbuffer(renderbuffer);

//gl.disableVertexAttribArray(vPos);
```



render1.js (10/13)

```
gl.useProgram(program2);

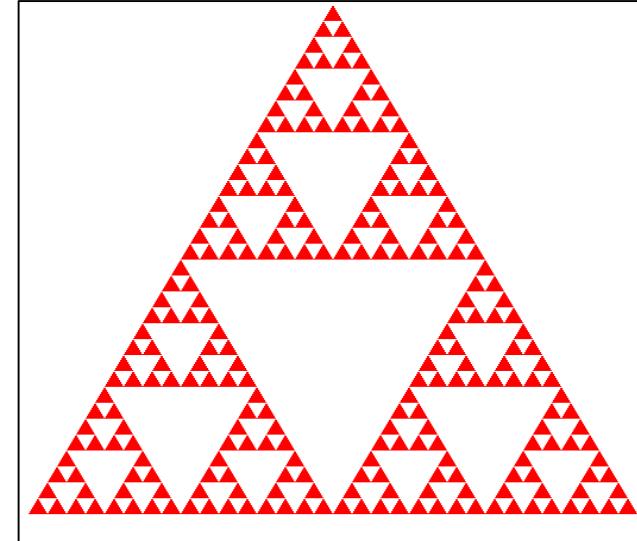
gl.activeTexture(gl.TEXTURE0);

gl.bindTexture(gl.TEXTURE_2D, texture1);

// send data to GPU for normal render

buffer2 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer2);
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);

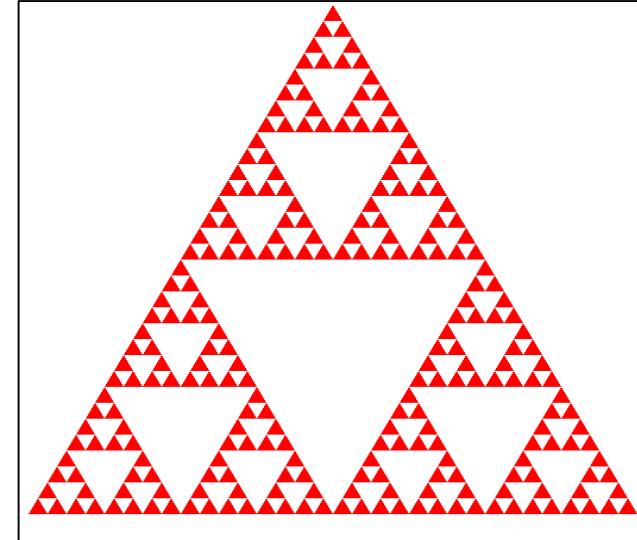
var vPosition = gl.getAttribLocation( program2, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
```



render1.js (11/13)

```
buffer3 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer3);
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoord), gl.STATIC_DRAW);

var vTexCoord = gl.getAttribLocation( program2, "vTexCoord");
gl.vertexAttribPointer( vTexCoord, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vTexCoord );
```



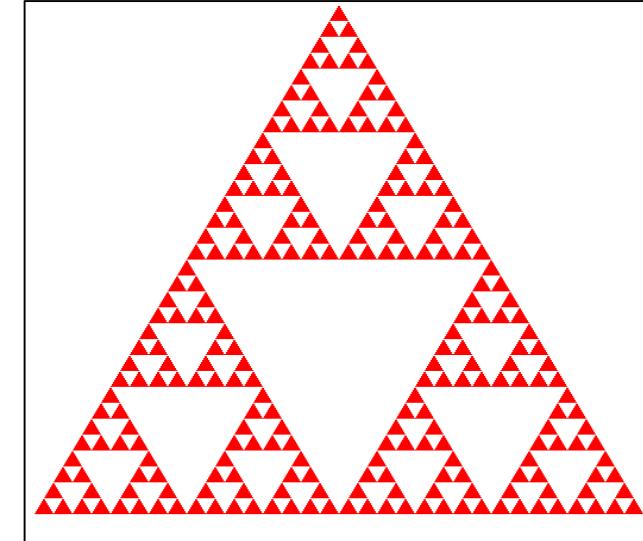
render1.js (12/13)

```
gl.uniform1i( gl.getUniformLocation(program2, "texture"), 0);

gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
gl.viewport(0, 0, 512, 512);

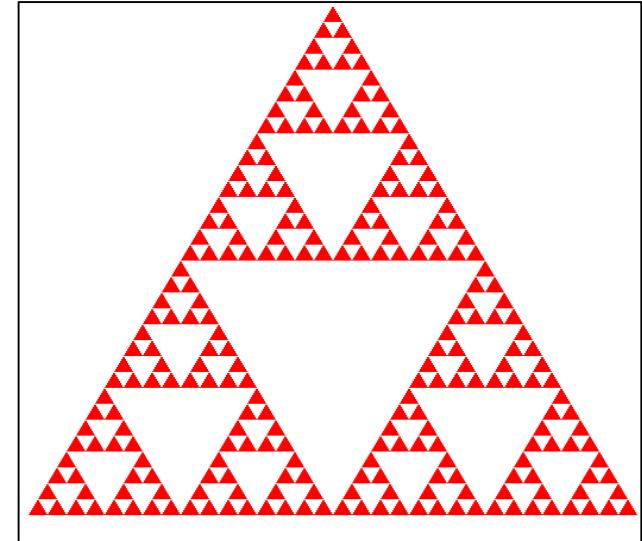
render();

} // end of window.onload
```

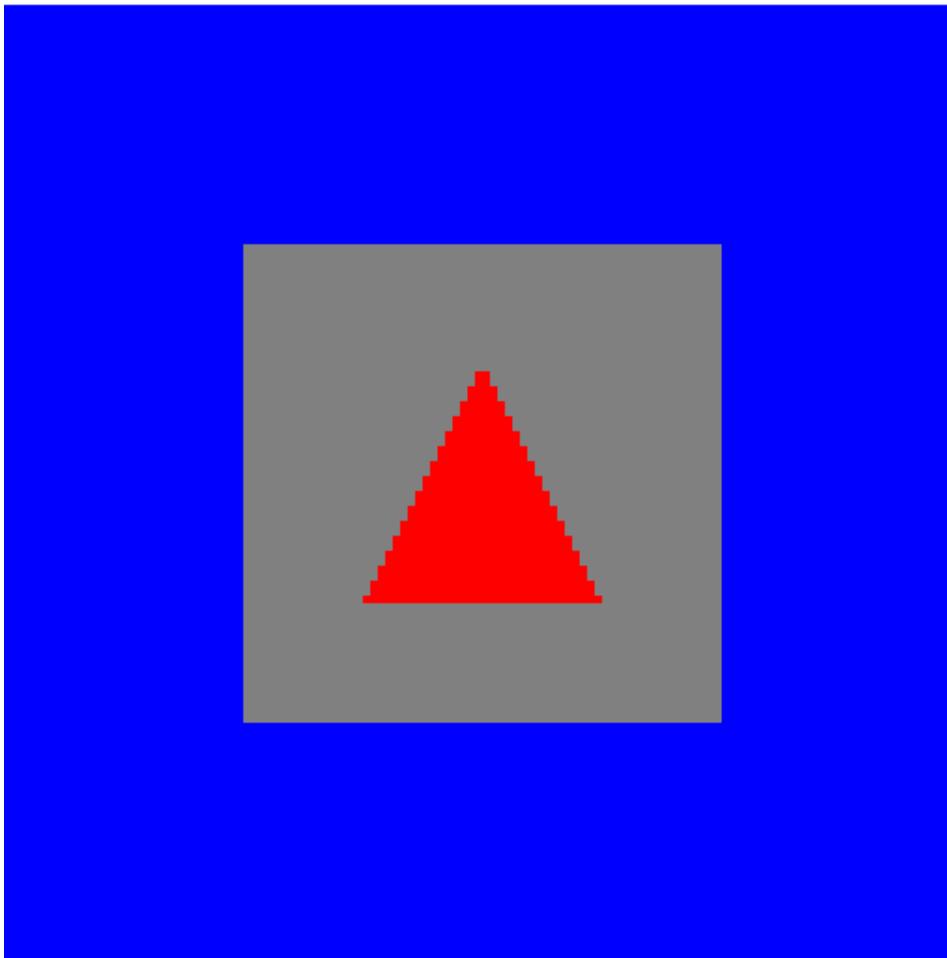


render1.js (13/13)

```
function render() {  
  
    gl.clear( gl.COLOR_BUFFER_BIT );  
    gl.drawArrays(gl.TRIANGLES, 0, 6);  
  
} // end of render()
```



Sample Programs: render1v2.html, render1v2.js



Render a triangle to a texture
and displays it by a second
rendering on a smaller square
so blue clear color is visible
around rendered square

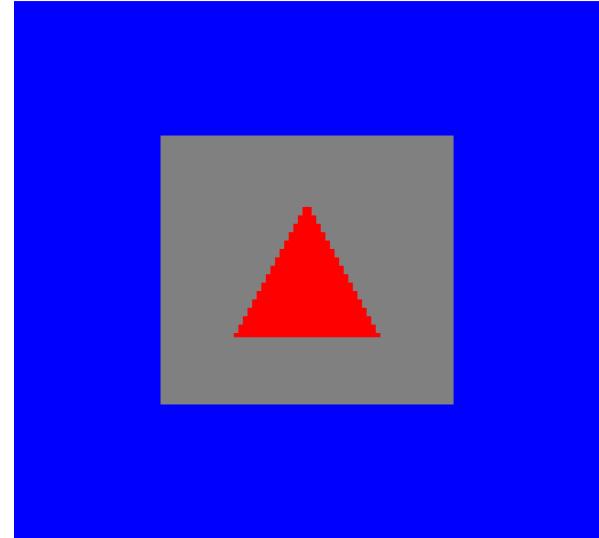
render1v2.html (1/5)

```
<!DOCTYPE html>
<html>

<script id="vertex-shader1" type="x-shader/x-vertex">

attribute vec4 vPosition;

void main()
{
    gl_Position = vPosition;
}
</script>
```



render1v2.html (2/5)

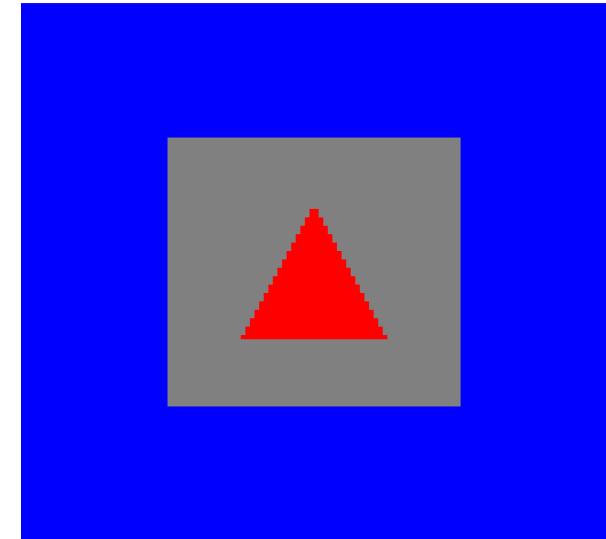
```
<script id="vertex-shader2" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec2 vTexCoord;

varying vec2 fTexCoord;

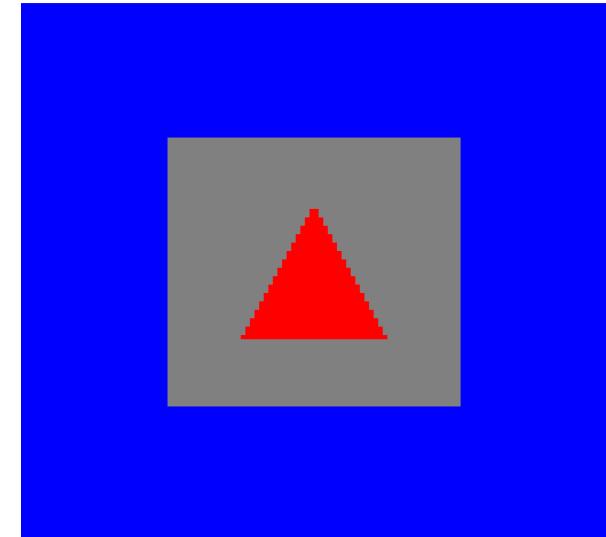
void main()
{
    gl_Position = vPosition;
    fTexCoord = vTexCoord;

}
```



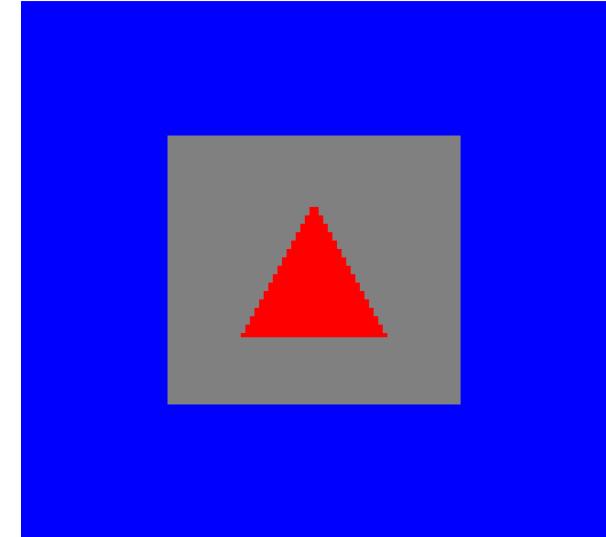
render1v2.html (3/5)

```
<script id="fragment-shader1" type="x-shader/x-fragment">  
  
precision mediump float;  
  
void  
main()  
{  
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);  
  
}  
</script>
```



render1v2.html (4/5)

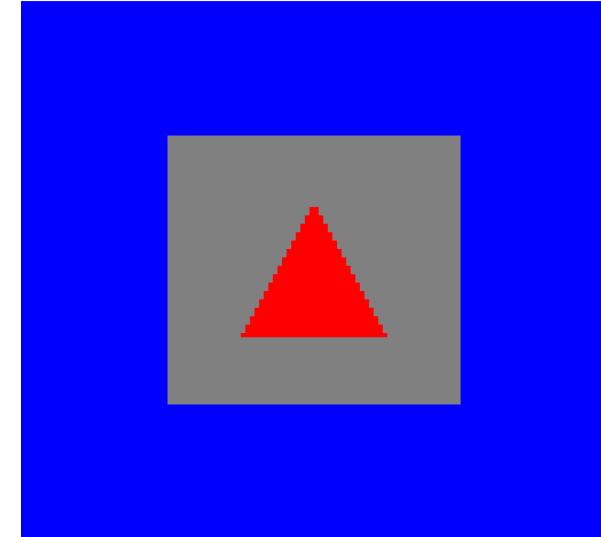
```
<script id="fragment-shader2" type="x-shader/x-fragment">  
  
precision mediump float;  
  
varying vec2 fTexCoord;  
  
uniform sampler2D texture;  
  
void main()  
{  
    gl_FragColor = texture2D( texture, fTexCoord);  
  
}  
</script>
```



render1v2.html (5/5)

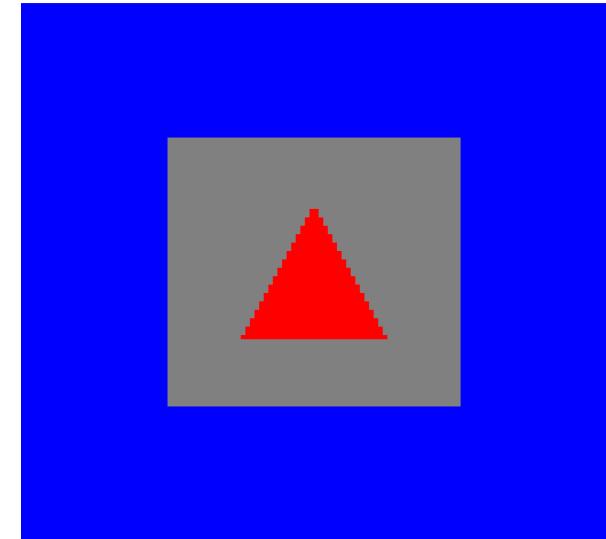
```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/InitShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="render1v2.js"></script>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



render1v2.js (1/10)

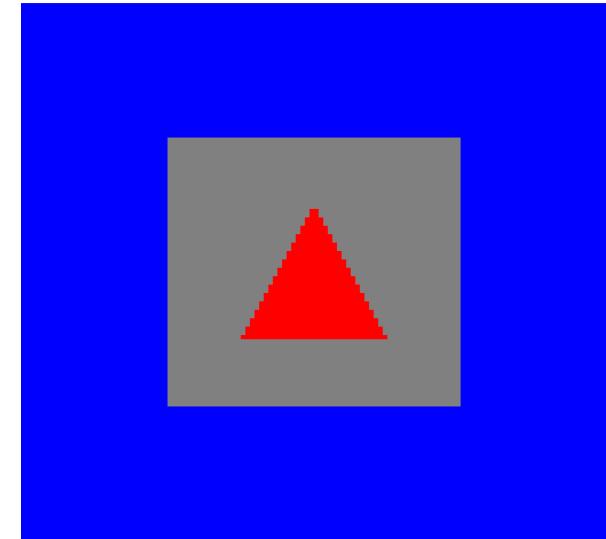
```
var canvas;  
var gl;  
  
// quad texture coordinates  
  
var texCoord = [  
    vec2(0, 0),  
    vec2(0, 1),  
    vec2(1, 1),  
    vec2(1, 1),  
    vec2(1, 0),  
    vec2(0, 0)  
];
```



render1v2.js (2/10)

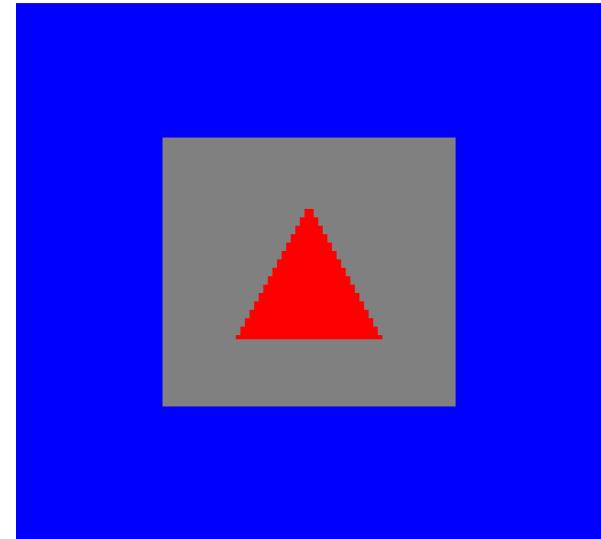
```
// quad vertices  
  
var vertices = [  
    vec2( -0.5, -0.5 ),  
    vec2( -0.5,  0.5 ),  
    vec2(  0.5,  0.5 ),  
    vec2(  0.5, -0.5 ),  
    vec2(  0.5, -0.5 ),  
    vec2( -0.5, -0.5 )  
];  
  
// triangle vertices
```

```
var pointsArray = [  
    vec2(-0.5, -0.5),  
    vec2( 0,  0.5),  
    vec2( 0.5, -0.5)  
];
```



render1v2.js (3/10)

```
var program1, program2;  
var texture1;  
  
var framebuffer, renderbuffer;  
  
window.onload = function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }
```

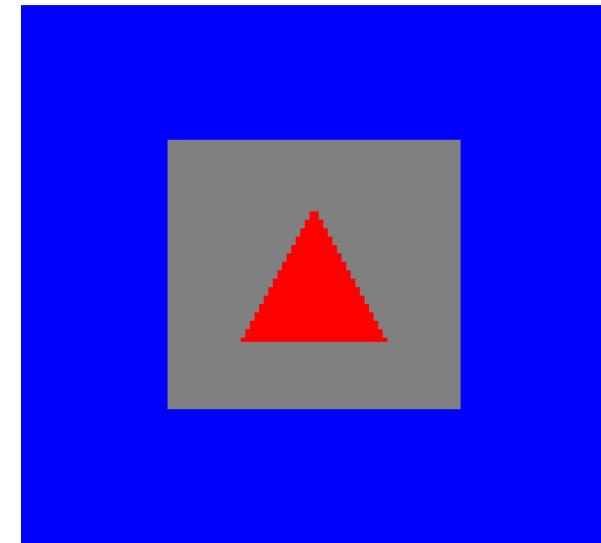


render1v2.js (4/10)

// Create an empty texture

```
texture1 = gl.createTexture();
gl.activeTexture( gl.TEXTURE0 );
gl.bindTexture( gl.TEXTURE_2D, texture1 );
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 64, 64, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);
gl.generateMipmap(gl.TEXTURE_2D);
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );

gl.bindTexture(gl.TEXTURE_2D, null);
```



render1v2.js (5/10)

```
// Allocate a frame buffer object
```

```
framebuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, framebuffer);
```

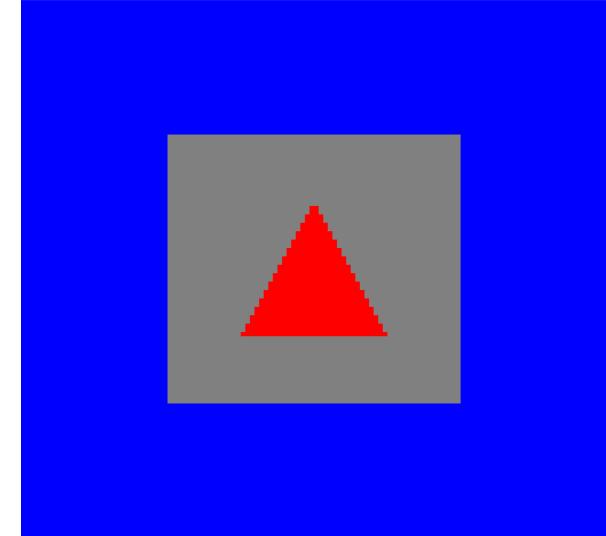
```
renderbuffer = gl.createRenderbuffer();
gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);
```

```
// Attach color buffer
```

```
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture1, 0);
```

```
// check for completeness
```

```
var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);
if(status != gl.FRAMEBUFFER_COMPLETE) alert('Frame Buffer Not Complete');
```



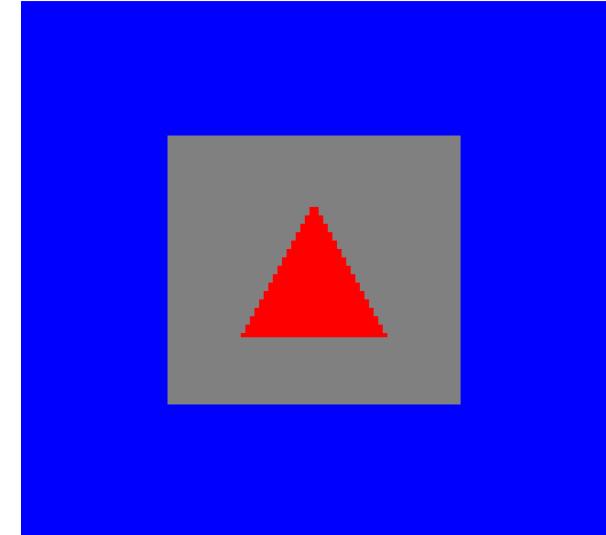
render1v2.js (6/10)

```
// Load shaders and initialize attribute buffers
program1 = initShaders( gl, "vertex-shader1", "fragment-shader1" );
program2 = initShaders( gl, "vertex-shader2", "fragment-shader2" );

gl.useProgram( program1 );

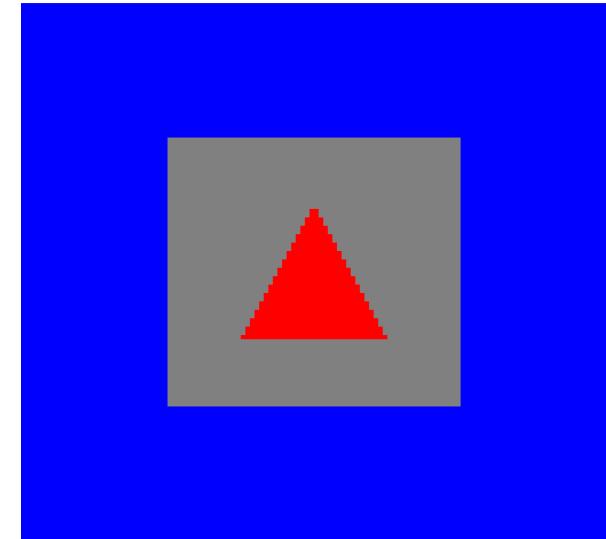
// Create and initialize a buffer object with triangle vertices
var buffer1 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer1 );
gl.bufferData( gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW );

// Initialize the vertex position attribute from the vertex shader
var vPosition = gl.getAttribLocation( program1, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
```



render1v2.js (7/10)

```
// Render one triangle  
gl.viewport(0, 0, 64, 64);  
gl.clearColor(0.5, 0.5, 0.5, 1.0);  
gl.clear(gl.COLOR_BUFFER_BIT );  
  
gl.drawArrays(gl.TRIANGLES, 0, 3);  
  
// Bind to window system frame buffer, unbind the texture  
gl.bindFramebuffer(gl.FRAMEBUFFER, null);  
gl.bindRenderbuffer(gl.RENDERBUFFER, null);  
  
gl.disableVertexAttribArray(vPosition);  
  
gl.useProgram(program2);  
  
gl.activeTexture(gl.TEXTURE0);  
gl.bindTexture(gl.TEXTURE_2D, texture1);
```

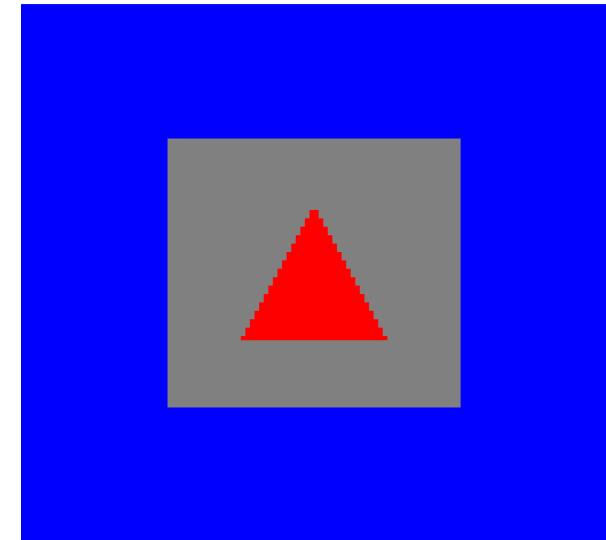


render1v2.js (8/10)

```
// send data to GPU for normal render

var buffer2 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer2);
gl.bufferData(gl.ARRAY_BUFFER,  new flatten(vertices), gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program2, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
```



render1v2.js (9/10)

```
var buffer3 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer3);
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoord), gl.STATIC_DRAW);

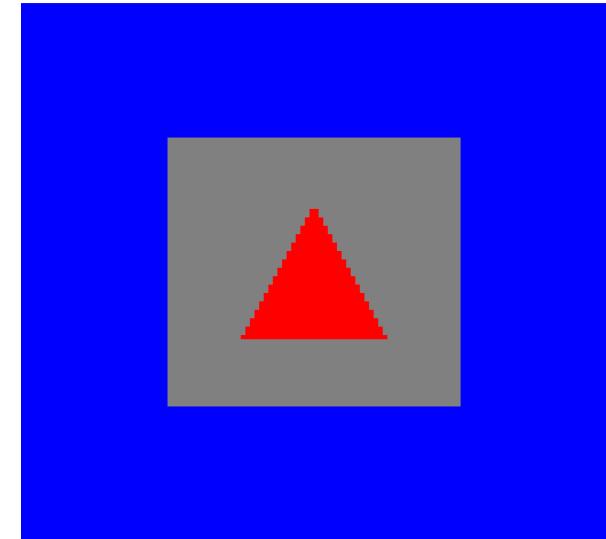
var vTexCoord = gl.getAttribLocation( program2, "vTexCoord");
gl.vertexAttribPointer( vTexCoord, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vTexCoord );

gl.uniform1i( gl.getUniformLocation(program2, "texture"), 0);

gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

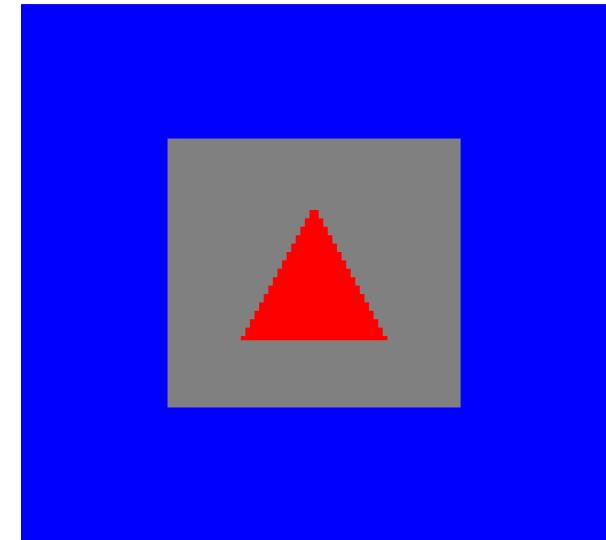
gl.viewport(0, 0, 512, 512);

render();
} // end of window.onload
```

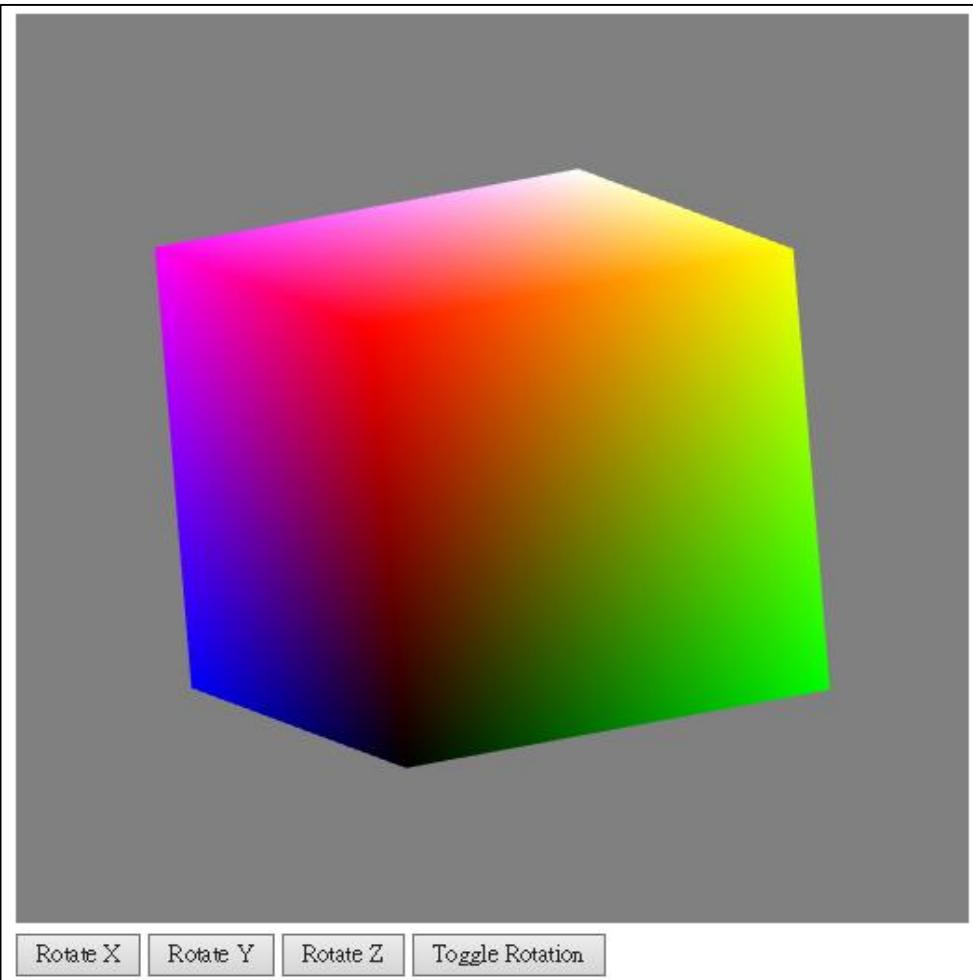
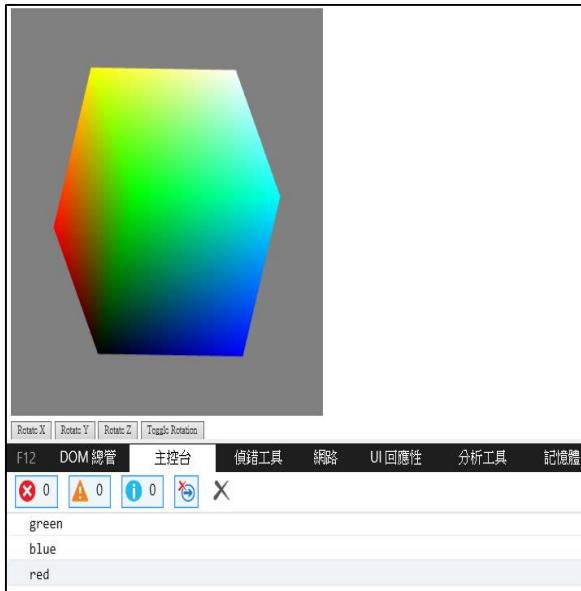


render1v2.js (10/10)

```
function render() {  
  
    gl.clearColor(0.0, 0.0, 1.0, 1.0);  
    gl.clear( gl.COLOR_BUFFER_BIT );  
  
    // render quad with texture  
  
    gl.drawArrays(gl.TRIANGLES, 0, 6);  
  
} // end of render()
```



Sample Programs: pickCube.html, pickCube.js



Rotating cube rendered off screen with solid colors for each face which are used to identify which face the mouse is clicked on. Color of face is logged onto the console.

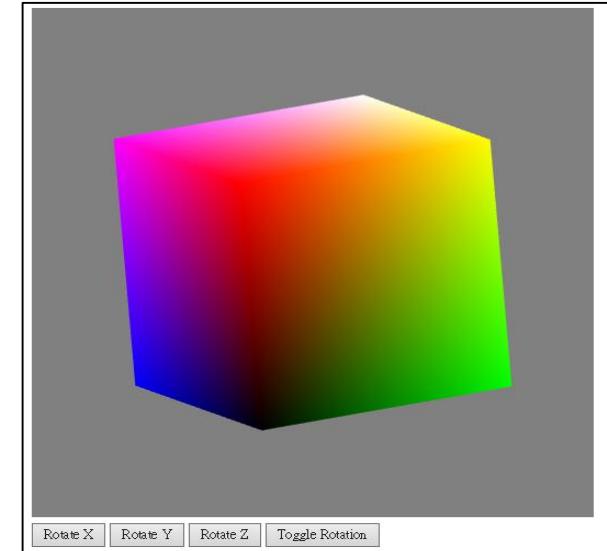
pickCube.html (1/5)

```
<!DOCTYPE html>
<html>
<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;

uniform vec3 theta;

void main()
{
    // Compute the sines and cosines of theta for each of
    // the three axes in one computation.
    vec3 angles = radians( theta );
    vec3 c = cos( angles );
    vec3 s = sin ( angles );
```



pickCube.html (2/5)

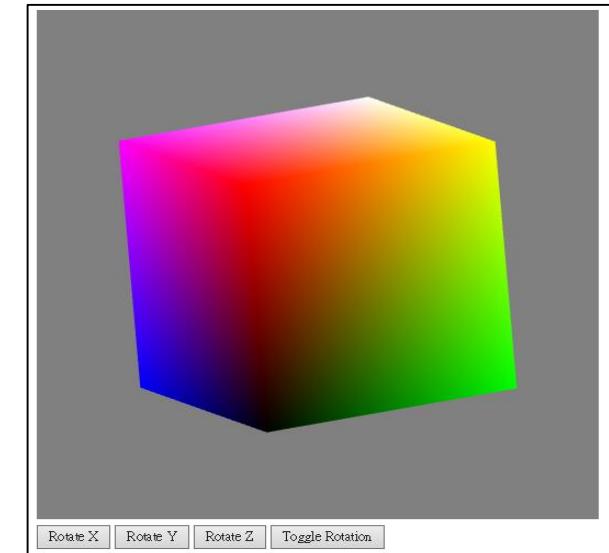
```
// Remember: these matrices are column-major
mat4 rx = mat4( 1.0, 0.0, 0.0, 0.0,
                  0.0, c.x, s.x, 0.0,
                  0.0, -s.x, c.x, 0.0,
                  0.0, 0.0, 0.0, 1.0 );

mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                  0.0, 1.0, 0.0, 0.0,
                  s.y, 0.0, c.y, 0.0,
                  0.0, 0.0, 0.0, 1.0 );

mat4 rz = mat4( c.z, -s.z, 0.0, 0.0,
                  s.z, c.z, 0.0, 0.0,
                  0.0, 0.0, 1.0, 0.0,
                  0.0, 0.0, 0.0, 1.0 );

fColor = vColor;
gl_Position = rz * ry * rx * vPosition;
}

</script>
```



pickCube.html (3/5)

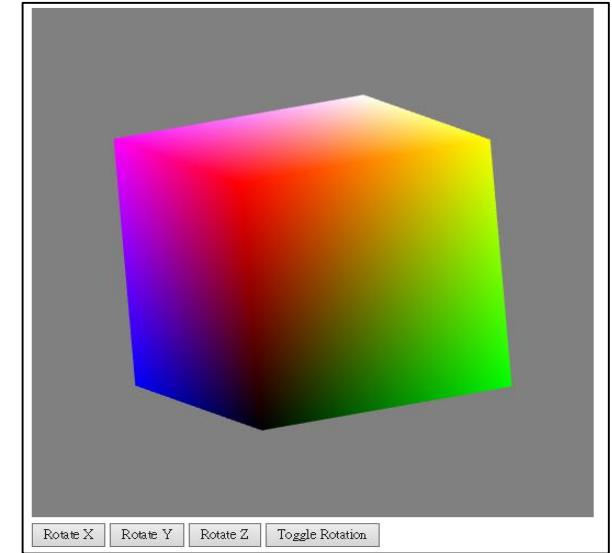
```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

uniform int i;

varying vec4 fColor;

void main()
{
    vec4 c[7];
    c[0] = fColor; ← normal rendering
    c[1] = vec4(1.0, 0.0, 0.0, 1.0);
    c[2] = vec4(0.0, 1.0, 0.0, 1.0);
    c[3] = vec4(0.0, 0.0, 1.0, 1.0);
    c[4] = vec4(1.0, 1.0, 0.0, 1.0);
    c[5] = vec4(0.0, 1.0, 1.0, 1.0);
    c[6] = vec4(1.0, 0.0, 1.0, 1.0); ← 6 colors for 6 faces of a cube: render to texture
}
```



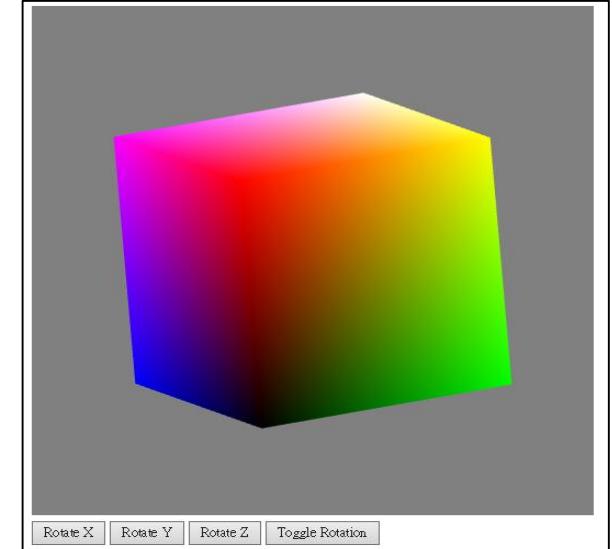
pickCube.html (4/5)

```
if (i==0) gl_FragColor = c[0];  
else if (i==1) gl_FragColor = c[1];  
else if (i==2) gl_FragColor = c[2];  
else if (i==3) gl_FragColor = c[3];  
else if (i==4) gl_FragColor = c[4];  
else if (i==5) gl_FragColor = c[5];  
else if (i==6) gl_FragColor = c[6];  
}  
</script>
```

```
<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="pickCube.js"></script>
```

normal rendering

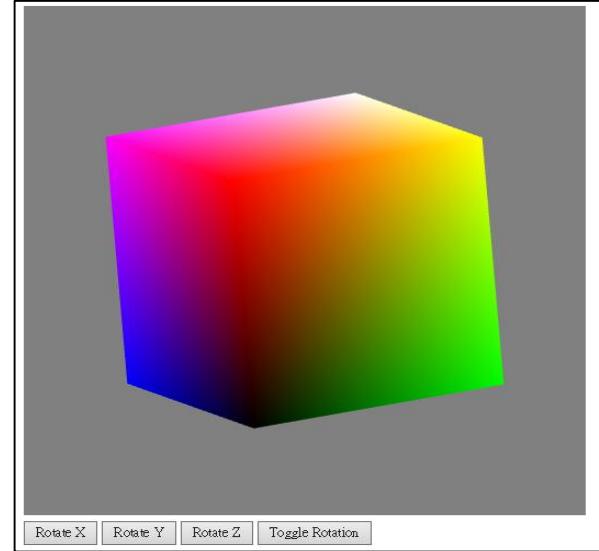
i (>0): render to texture
with base colors



pickCube.html (5/5)

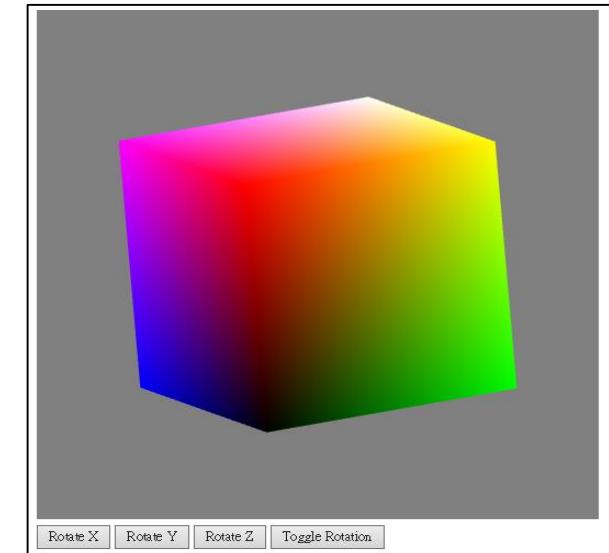
```
<body>
<div>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</div>
<div>
<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
<button id = "ButtonT">Toggle Rotation</button>
</div>

</body>
```



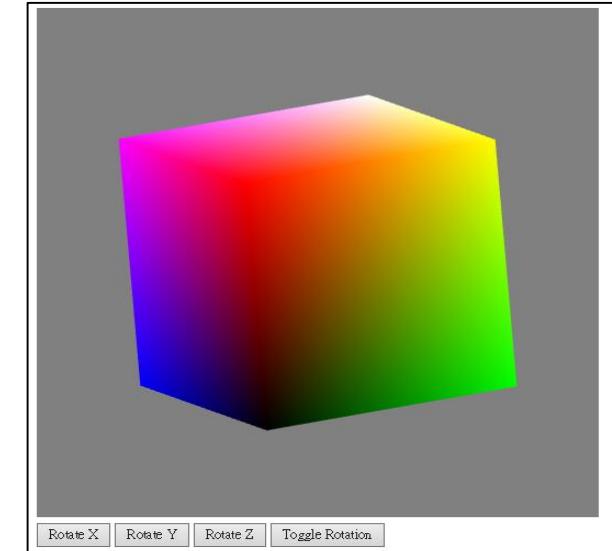
pickCube.js (1/14)

```
var canvas;  
var gl;  
  
var program;  
  
var NumVertices = 36;  
  
var pointsArray = [];  
var colorsArray = [];  
  
var framebuffer;  
  
var flag = true;  
  
var color = new Uint8Array(4);
```



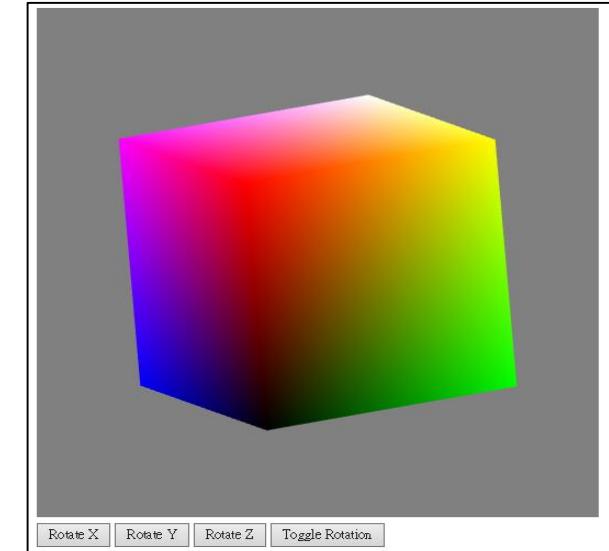
pickCube.js (2/14)

```
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 ),  
];
```



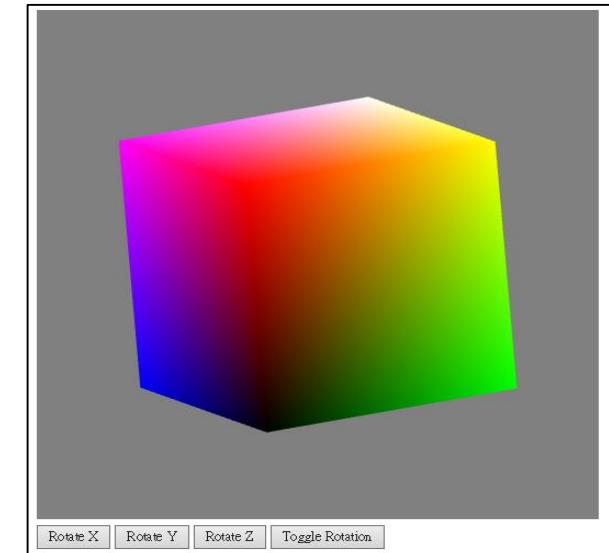
pickCube.js (3/14)

```
var vertexColors = [  
    vec4( 0.0, 0.0, 0.0, 1.0 ), // black  
    vec4( 1.0, 0.0, 0.0, 1.0 ), // red  
    vec4( 1.0, 1.0, 0.0, 1.0 ), // yellow  
    vec4( 0.0, 1.0, 0.0, 1.0 ), // green  
    vec4( 0.0, 0.0, 1.0, 1.0 ), // blue  
    vec4( 1.0, 0.0, 1.0, 1.0 ), // magenta  
    vec4( 1.0, 1.0, 1.0, 1.0 ), // white  
    vec4( 0.0, 1.0, 1.0, 1.0 ), // cyan  
];
```



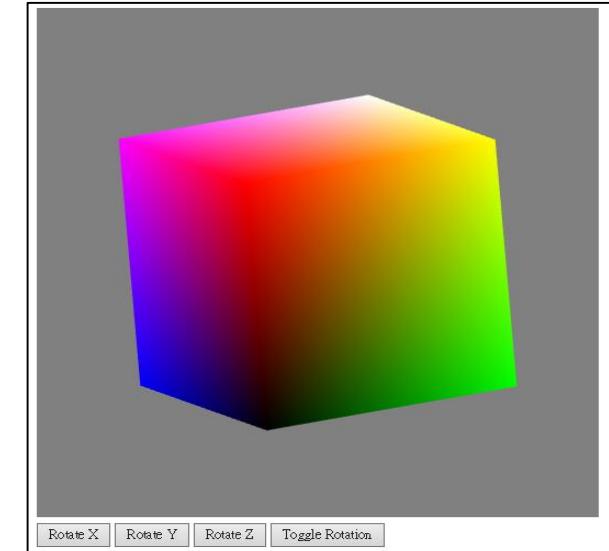
pickCube.js (4/14)

```
var xAxis = 0;  
var yAxis = 1;  
var zAxis = 2;  
var axis = xAxis;  
  
var theta = [45.0, 45.0, 45.0];  
  
var thetaLoc;  
  
var Index = 0;
```



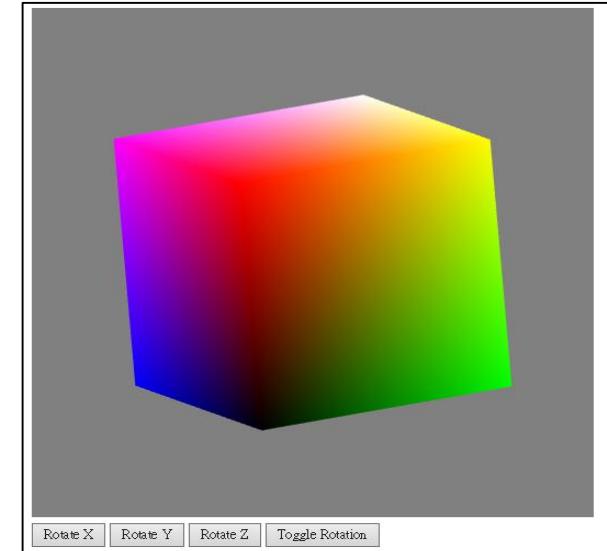
pickCube.js (5/14)

```
function quad(a, b, c, d) {  
    pointsArray.push(vertices[a]);  
    colorsArray.push(vertexColors[a]);  
    pointsArray.push(vertices[b]);  
    colorsArray.push(vertexColors[b]);  
    pointsArray.push(vertices[c]);  
    colorsArray.push(vertexColors[c]);  
    pointsArray.push(vertices[a]);  
    colorsArray.push(vertexColors[a]);  
    pointsArray.push(vertices[c]);  
    colorsArray.push(vertexColors[c]);  
    pointsArray.push(vertices[d]);  
    colorsArray.push(vertexColors[d]);  
}
```



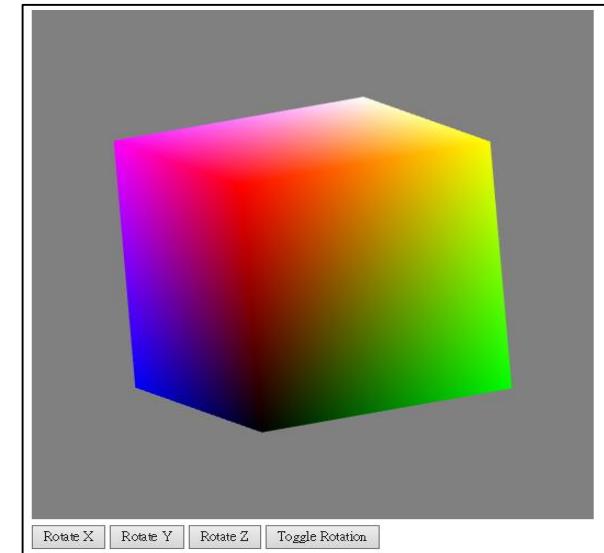
pickCube.js (6/14)

```
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 );
}
```



pickCube.js (7/14)

```
window.onload = function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    var ctx = canvas.getContext("experimental-webgl", {preserveDrawingBuffer: true});  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 0.5, 0.5, 0.5, 1.0 );  
  
    gl.enable(gl.CULL_FACE);
```



pickCube.js (8/14)

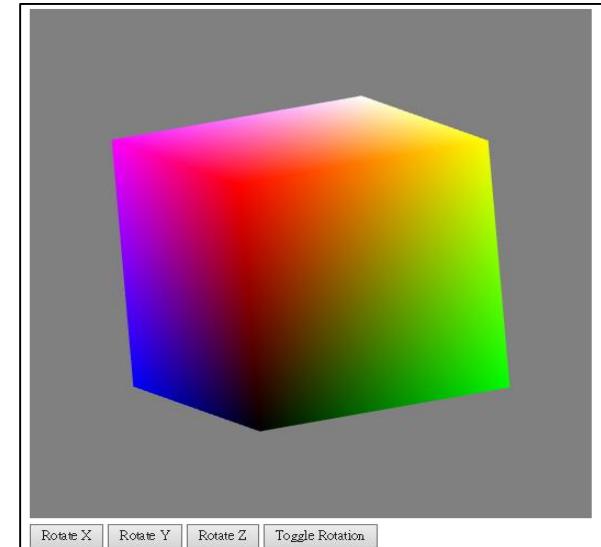
```
var texture = gl.createTexture();
gl.bindTexture( gl.TEXTURE_2D, texture );
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 512, 512, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);
gl.generateMipmap(gl.TEXTURE_2D);

// Allocate a frame buffer object

framebuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, framebuffer);

// Attach color buffer

gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);
```



pickCube.js (9/14)

```
// check for completeness

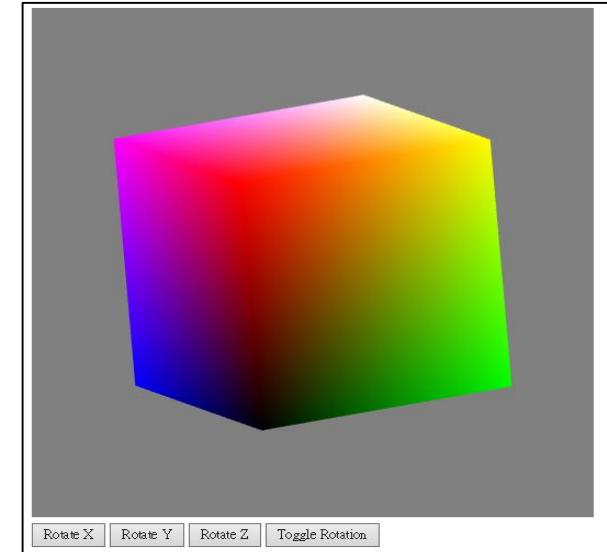
//var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);
//if(status != gl.FRAMEBUFFER_COMPLETE) alert('Frame Buffer Not Complete');

gl.bindFramebuffer(gl.FRAMEBUFFER, null);

//
// Load shaders and initialize attribute buffers
//

program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

colorCube();
```



pickCube.js (10/14)

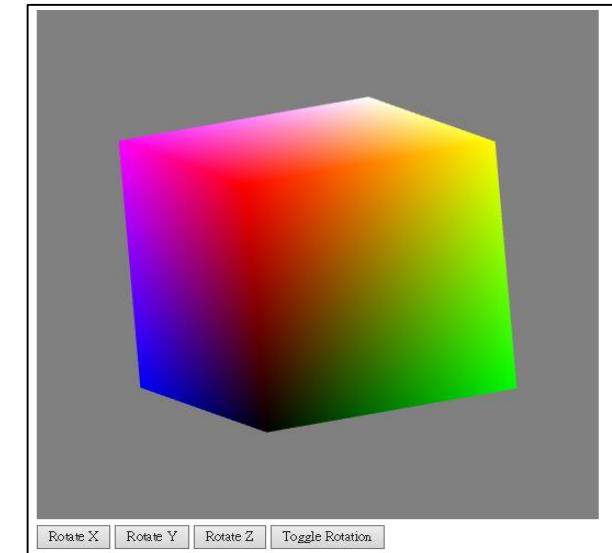
```
var cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(colorsArray), gl.STATIC_DRAW );

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(pointsArray), gl.STATIC_DRAW );

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

thetaLoc = gl.getUniformLocation(program, "theta");
```



pickCube.js (11/14)

```
document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};
document.getElementById("ButtonT").onclick = function() {flag = !flag};

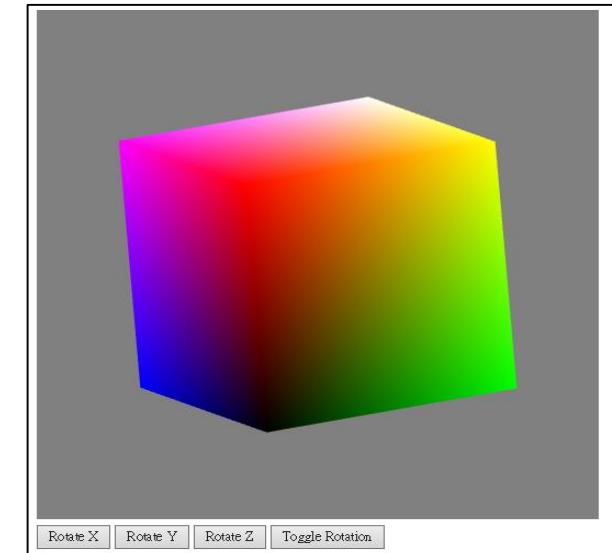
canvas.addEventListener("mousedown", function() {

gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
gl.clear( gl.COLOR_BUFFER_BIT);
gl.uniform3fv(thetaLoc, theta);
for(var i=0; i<6; i++) {
    gl.uniform1i(gl.getUniformLocation(program, "i"), i+1);
    gl.drawArrays( gl.TRIANGLES, 6*i, 6 );
}
})
```

i+1 (>0): render to texture



Draw face i (2 triangles with 6 vertices).
Each face has different color. (color[i+1])



pickCube.js (12/14)

```
var x = event.clientX;  
var y = canvas.height - event.clientY;
```

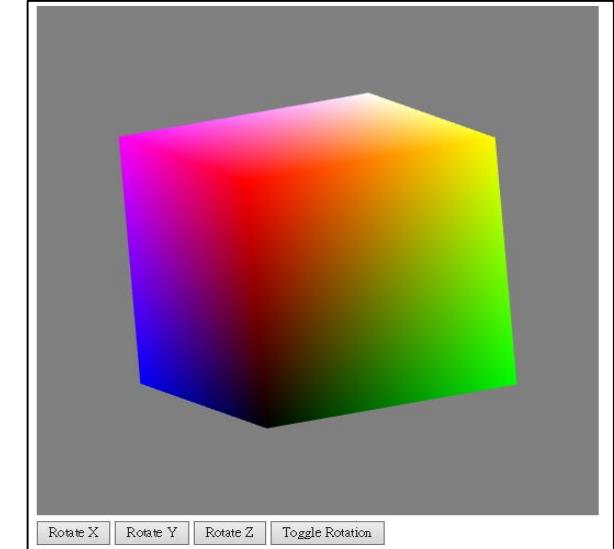
Get mouse position

```
gl.readPixels(x, y, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, color);
```

```
if (color[0]==255)  
    if (color[1]==255) console.log("yellow");  
    else if (color[2]==255) console.log("magenta");  
    else console.log("red");  
else if (color[1]==255)  
    if (color[2]==255) console.log("cyan");  
    else console.log("green");  
    else if(color[2]==255) console.log("blue");  
    else console.log("background");  
gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

Get color with mouse location and output it

Find which color was read and output it

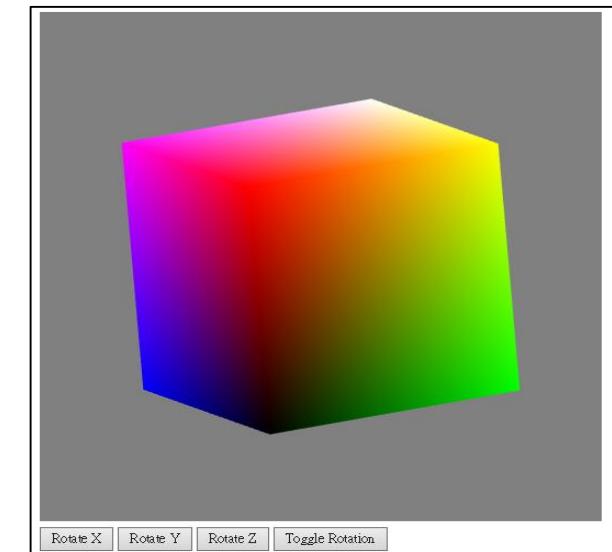


pickCube.js (13/14)

```
gl.uniform1i(gl.getUniformLocation(program, "i"), 0);
gl.clear( gl.COLOR_BUFFER_BIT );
gl.uniform3fv(thetaLoc, theta);
gl.drawArrays(gl.TRIANGLES, 0, 36); // 36 vertices
}); // end of canvas.addEventListener("mousedown", function() {})

render();
} // end of window.onload
```

0: normal rendering



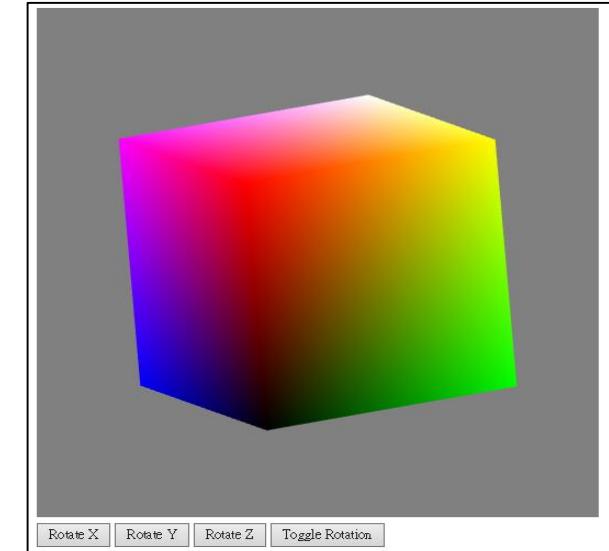
A cube has 6 faces and each face has 2 triangles.
Total vertices = 6 faces x 2 triangles/face x 3 vertices/triangle
= 36 vertices

pickCube.js (14/14)

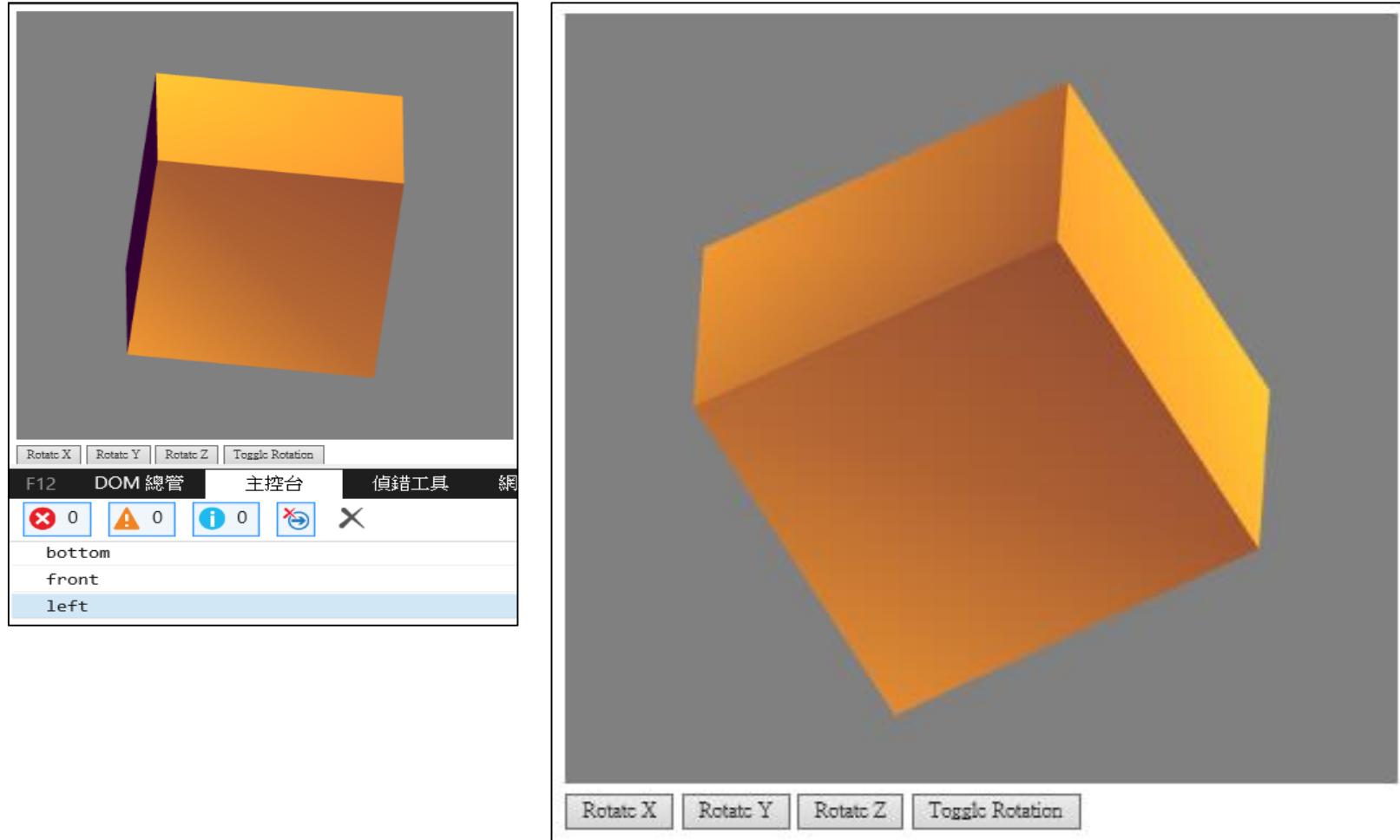
```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    if(flag) theta[axis] += 2.0;
    gl.uniform3fv(thetaLoc, theta);
    gl.uniform1i(gl.getUniformLocation(program, "i"), 0);
    gl.drawArrays( gl.TRIANGLES, 0, 36 );
    requestAnimFrame(render);
}
```

0: normal rendering

A cube has 6 faces and each face has 2 triangles.
Total vertices = 6 faces x 2 triangles/face x 3 vertices/triangle
= 36 vertices



Sample Programs: pickCube2.html, pickCube2.js



Rotating cube with lighting.
When mouse is clicked, the
face name (front, back,
right, left, top, bottom,
background) is logged onto
the console.

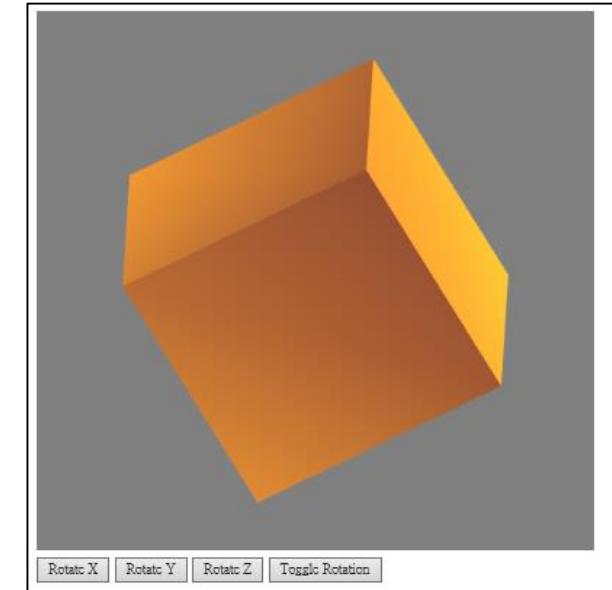
pickCube2.html (1/6)

```
<!DOCTYPE html>
<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="pickCube2.js"></script>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec3 vNormal;
varying vec4 fColor;

uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float shininess;
```



pickCube2.html (2/6)

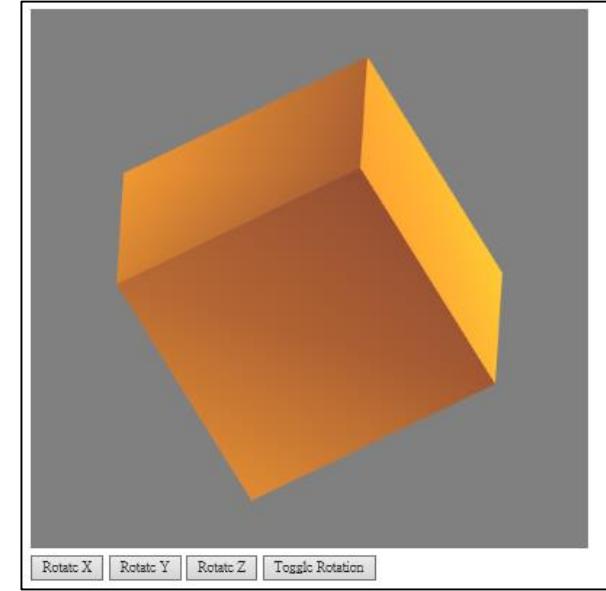
```
void main()
{
    vec3 pos = -(modelViewMatrix * vPosition).xyz;
    vec3 light = lightPosition.xyz;
    vec3 L    = normalize( light - pos );

    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E );

    vec4 NN = vec4(vNormal,0);

    // Transform vertex normal into eye coordinates

    vec3 N = normalize( (modelViewMatrix*NN).xyz);
```



pickCube2.html (3/6)

```
// Compute terms in the illumination equation
vec4 ambient = ambientProduct;

float Kd = max( dot(L, N), 0.0 );
vec4 diffuse = Kd*diffuseProduct;

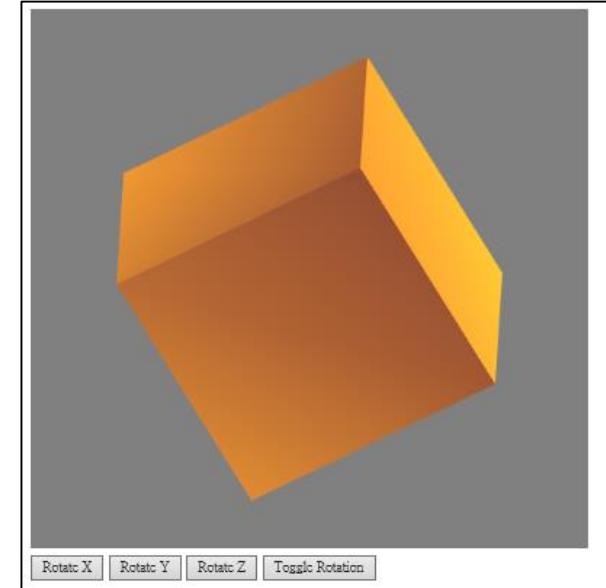
float Ks = pow( max(dot(N, H), 0.0), shininess );
vec4 specular = Ks * specularProduct;

if( dot(L, N) < 0.0 ) { specular = vec4(0.0, 0.0, 0.0, 1.0); }

gl_Position = projectionMatrix * modelViewMatrix * vPosition;
fColor = ambient + diffuse +specular;

fColor.a = 1.0;
}

</script>
```



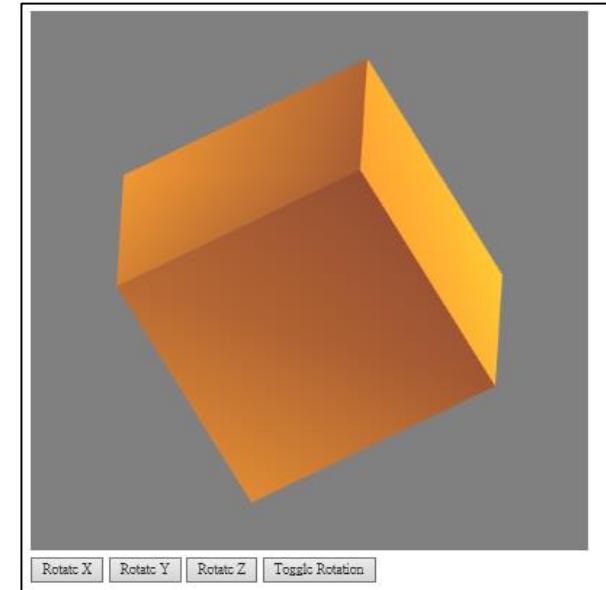
pickCube2.html (4/6)

```
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

uniform int i;

varying vec4 fColor;

void main()
{
    vec4 c[7];
    c[0] = fColor; ← normal rendering
    c[1] = vec4(1.0, 0.0, 0.0, 1.0);
    c[2] = vec4(0.0, 1.0, 0.0, 1.0);
    c[3] = vec4(0.0, 0.0, 1.0, 1.0);
    c[4] = vec4(1.0, 1.0, 0.0, 1.0); ← 6 colors for 6 faces of a cube: render to texture
    c[5] = vec4(0.0, 1.0, 1.0, 1.0);
    c[6] = vec4(1.0, 0.0, 1.0, 1.0);
```

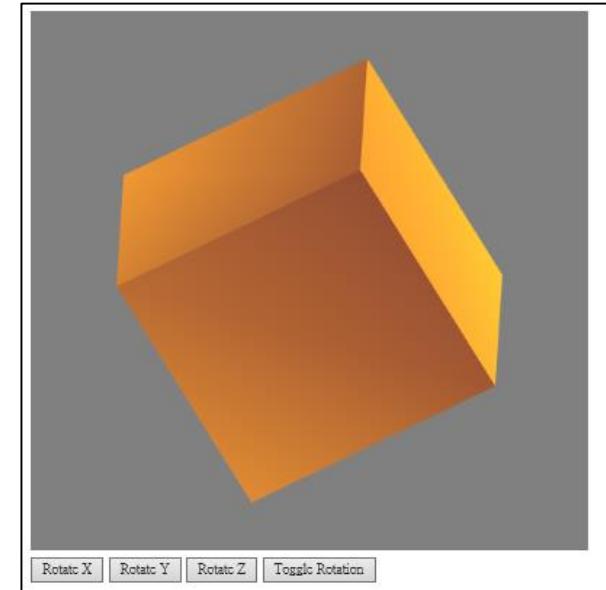


pickCube2.html (5/6)

```
if (i==0) gl_FragColor = c[0];  
else if (i==1) gl_FragColor = c[1];  
else if (i==2) gl_FragColor = c[2];  
else if (i==3) gl_FragColor = c[3];  
else if (i==4) gl_FragColor = c[4];  
else if (i==5) gl_FragColor = c[5];  
else if (i==6) gl_FragColor = c[6];  
}  
</script>
```

normal rendering

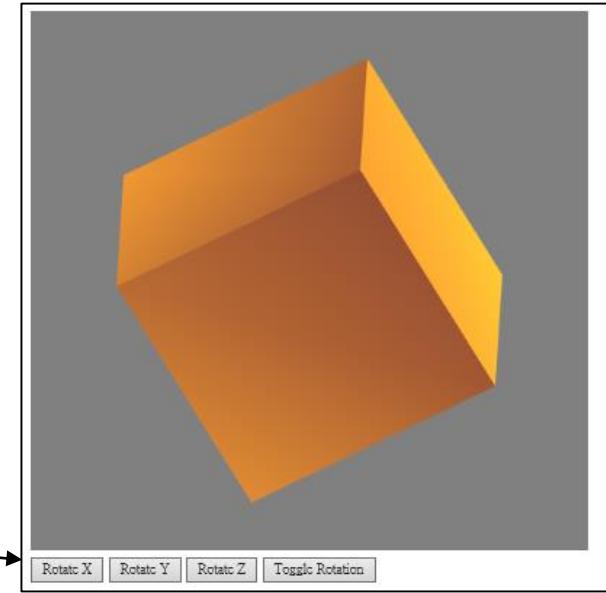
i (>0): render to texture
with base colors



pickCube2.html (6/6)

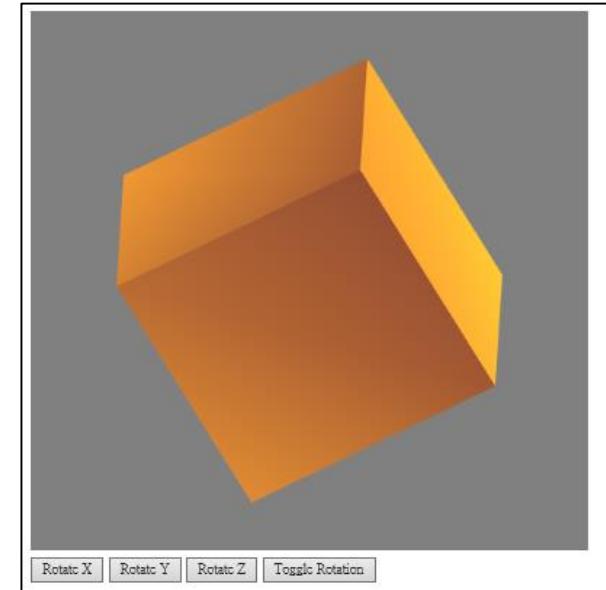
```
<body>
<div>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</div>
<div>
<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
<button id = "ButtonT">Toggle Rotation</button>
</div>

</body>
</html>
```



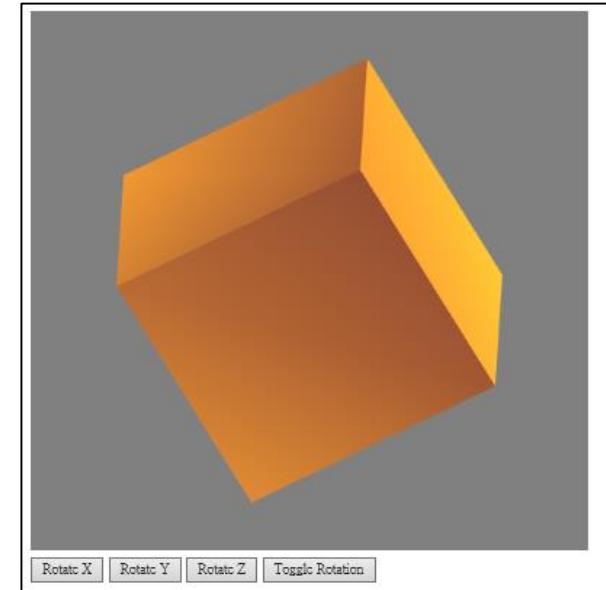
pickCube2.js (1/15)

```
var canvas;  
var gl;  
  
var program;  
  
var NumVertices = 36;  
  
var pointsArray = [];  
var normalsArray = [];  
  
var framebuffer;  
  
var flag = false;  
  
var color = new Uint8Array(4);
```



pickCube2.js (2/15)

```
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 ),  
];
```

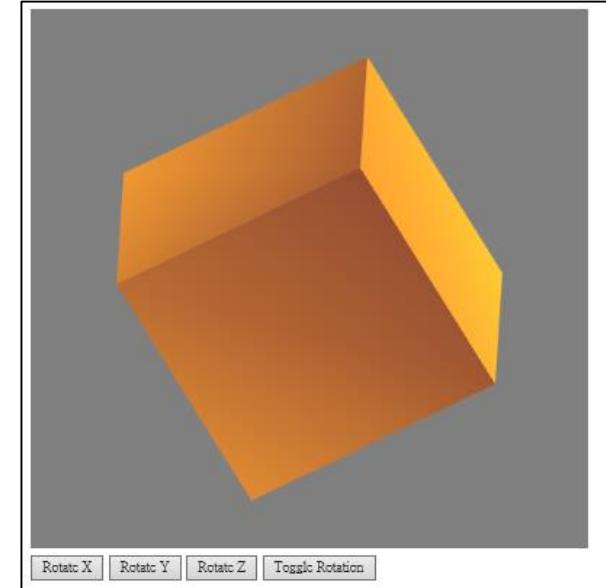


pickCube2.js (3/15)

```
var lightPosition = vec4(1.0, 1.0, 1.0, 0.0 );
var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0 );

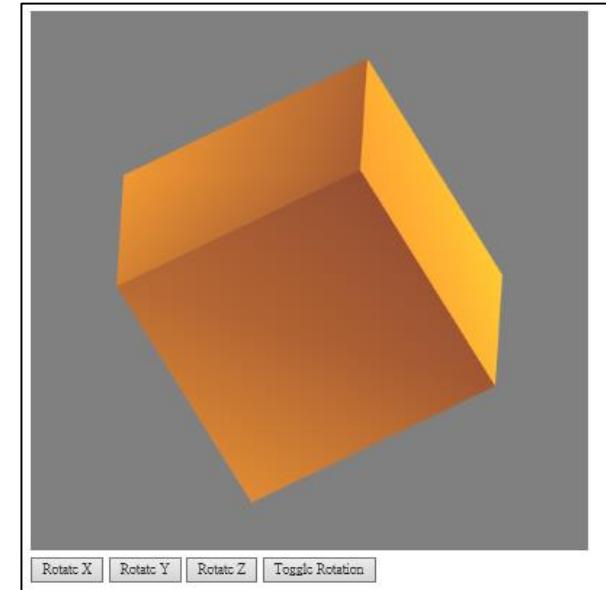
var materialAmbient = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialSpecular = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;

var ctm;
var ambientColor, diffuseColor, specularColor;
var modelView, projection;
var viewerPos;
var program;
```



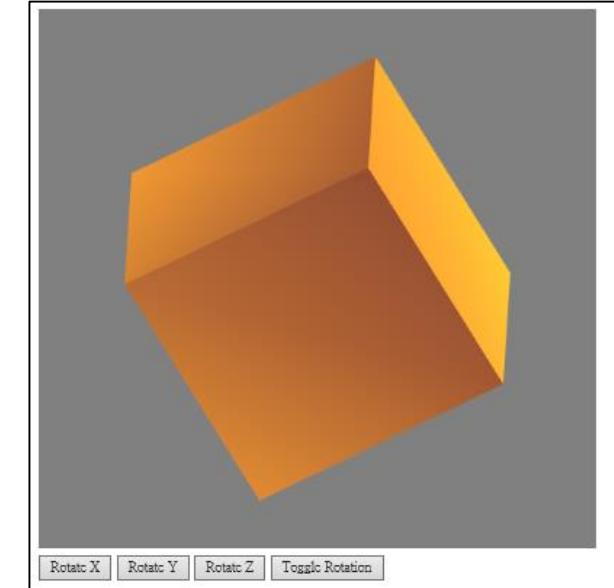
pickCube2.js (4/15)

```
var xAxis = 0;  
var yAxis = 1;  
var zAxis = 2;  
var axis = xAxis;  
  
var theta = [45.0, 45.0, 45.0];  
  
var thetaLoc;  
  
var Index = 0;
```



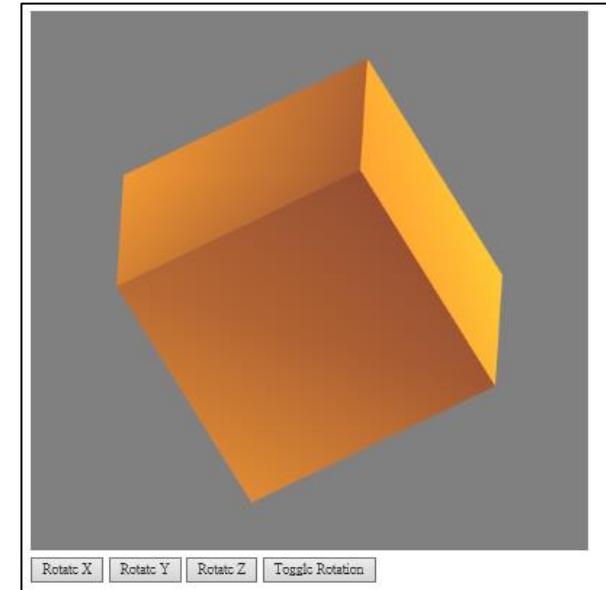
pickCube2.js (5/15)

```
function quad(a, b, c, d) {  
    var t1 = subtract(vertices[b], vertices[a]);  
    var t2 = subtract(vertices[c], vertices[b]);  
    var normal = cross(t1, t2);  
    var normal = vec3(normal);  
    normal = normalize(normal);  
    pointsArray.push(vertices[a]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[b]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[c]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[a]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[c]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[d]);  
    normalsArray.push(normal);  
}
```



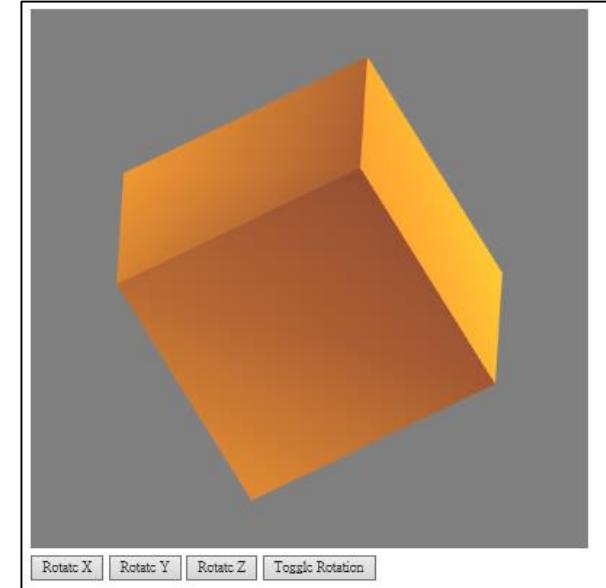
pickCube2.js (6/15)

```
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 );
}
```



pickCube2.js (7/15)

```
window.onload = function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    var ctx = canvas.getContext("experimental-webgl", {preserveDrawingBuffer: true});  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 0.5, 0.5, 0.5, 1.0 );  
  
    gl.enable(gl.CULL_FACE);  
  
    var texture = gl.createTexture();  
    gl.bindTexture( gl.TEXTURE_2D, texture );  
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);  
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 512, 512, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);  
    gl.generateMipmap(gl.TEXTURE_2D);
```



pickCube2.js (8/15)

```
// Allocate a frame buffer object
```

```
framebuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, framebuffer);
```

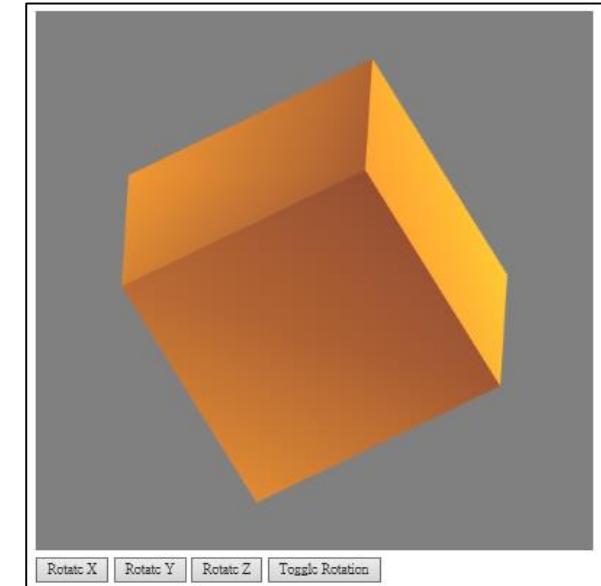
```
// Attach color buffer
```

```
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);
```

```
// check for completeness
```

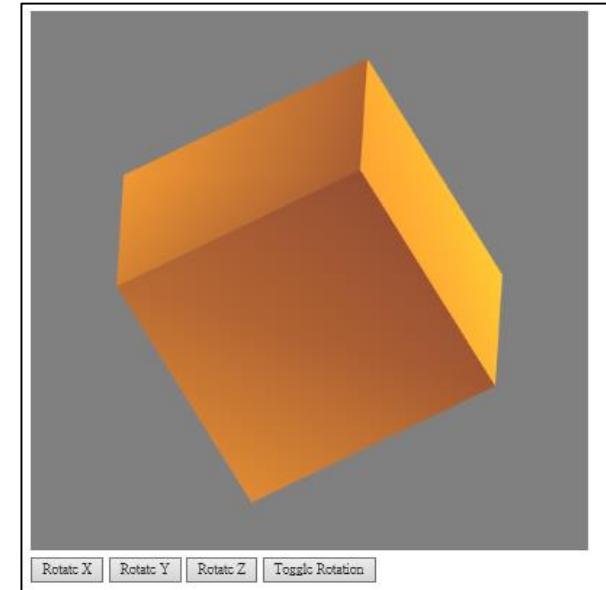
```
//var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);
//if(status != gl.FRAMEBUFFER_COMPLETE) alert('Frame Buffer Not Complete');
```

```
gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```



pickCube2.js (9/15)

```
//  
// Load shaders and initialize attribute buffers  
  
program = initShaders( gl, "vertex-shader", "fragment-shader" );  
gl.useProgram( program );  
  
colorCube();  
  
var nBuffer = gl.createBuffer();  
gl.bindBuffer( gl.ARRAY_BUFFER, nBuffer );  
gl.bufferData( gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW );  
  
var vNormal = gl.getAttribLocation( program, "vNormal" );  
gl.vertexAttribPointer( vNormal, 3, gl.FLOAT, false, 0, 0 );  
gl.enableVertexAttribArray( vNormal );
```



pickCube2.js (10/15)

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

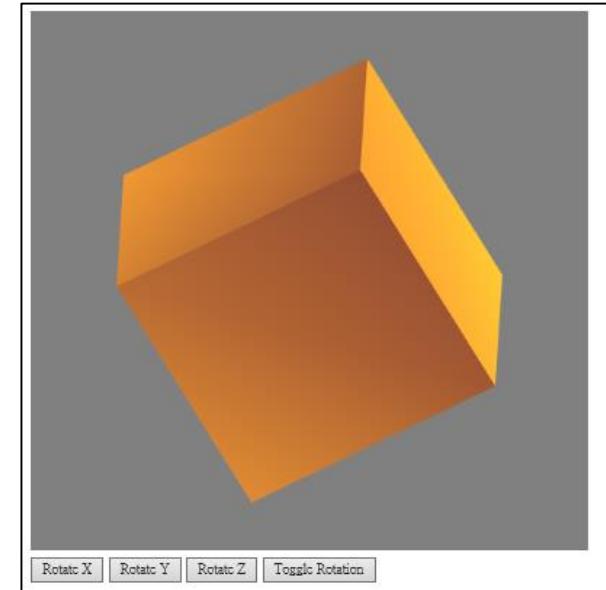
var vPosition = gl.getAttribLocation(program, "vPosition");
gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);

thetaLoc = gl.getUniformLocation(program, "theta");

viewerPos = vec3(0.0, 0.0, -20.0 );

projection = ortho(-1, 1, -1, 1, -100, 100);

ambientProduct = mult(lightAmbient, materialAmbient);
diffuseProduct = mult(lightDiffuse, materialDiffuse);
specularProduct = mult(lightSpecular, materialSpecular);
```



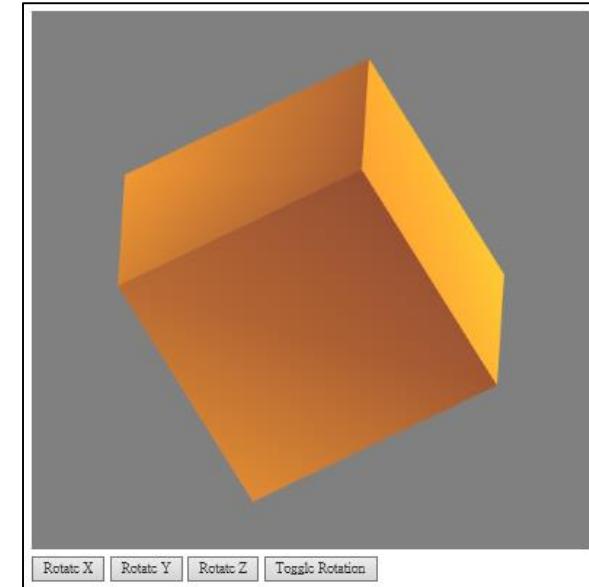
pickCube2.js (11/15)

```
document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};
document.getElementById("ButtonT").onclick = function() {flag = !flag};
```

```
gl.uniform4fv(gl.getUniformLocation(program, "ambientProduct"), flatten(ambientProduct) );
gl.uniform4fv(gl.getUniformLocation(program, "diffuseProduct"), flatten(diffuseProduct) );
gl.uniform4fv(gl.getUniformLocation(program, "specularProduct"), flatten(specularProduct) );
gl.uniform4fv(gl.getUniformLocation(program, "lightPosition"), flatten(lightPosition) );

gl.uniform1f(gl.getUniformLocation(program, "shininess"), materialShininess);

gl.uniformMatrix4fv( gl.getUniformLocation(program, "projectionMatrix"), false, flatten(projection));
```

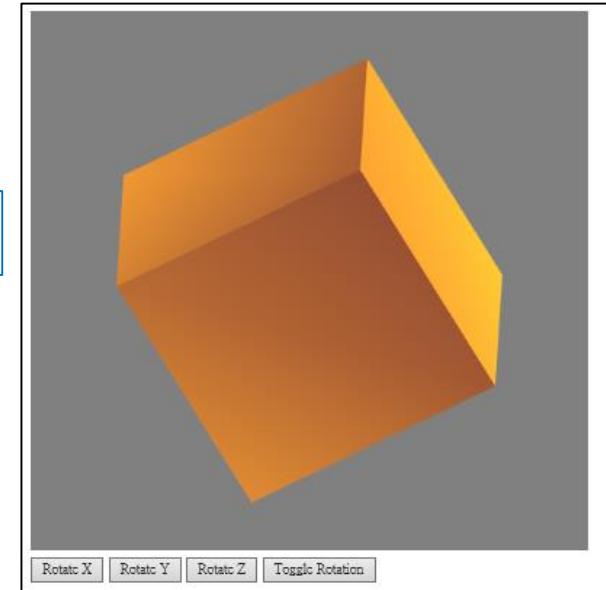


pickCube2.js (12/15)

```
canvas.addEventListener("mousedown", function() {  
  
    gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);  
    gl.clear( gl.COLOR_BUFFER_BIT);  
    gl.uniform3fv(thetaLoc, theta);  
    for(var i=0; i<6; i++) {  
        gl.uniform1i(gl.getUniformLocation(program, "i"), i+1);  
        gl.drawArrays( gl.TRIANGLES, 6*i, 6 );  
    }  
})
```

i+1 (>0): render to texture

Draw face i (2 triangles with 6 vertices).
Each face has different color. (color[i+1])



pickCube2.js (13/15)

```
var x = event.clientX;  
var y = canvas.height - event.clientY;
```

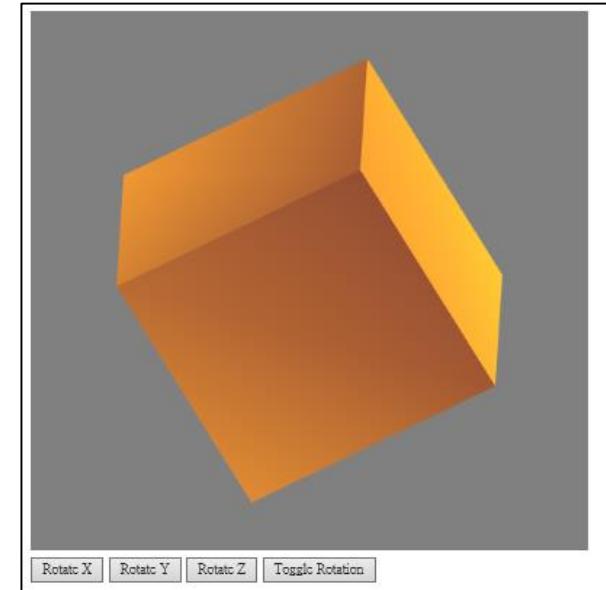
Get mouse position

```
gl.readPixels(x, y, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, color);
```

```
if (color[0]==255)  
    if (color[1]==255) console.log("front");  
    else if (color[2]==255) console.log("back");  
        else console.log("right");  
else if (color[1]==255)  
    if (color[2]==255) console.log("left");  
    else console.log("top");  
    else if(color[2]==255) console.log("bottom");  
        else console.log("background");  
gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

Get color with mouse location and output it

Find which face was read and output it



pickCube2.js (14/15)

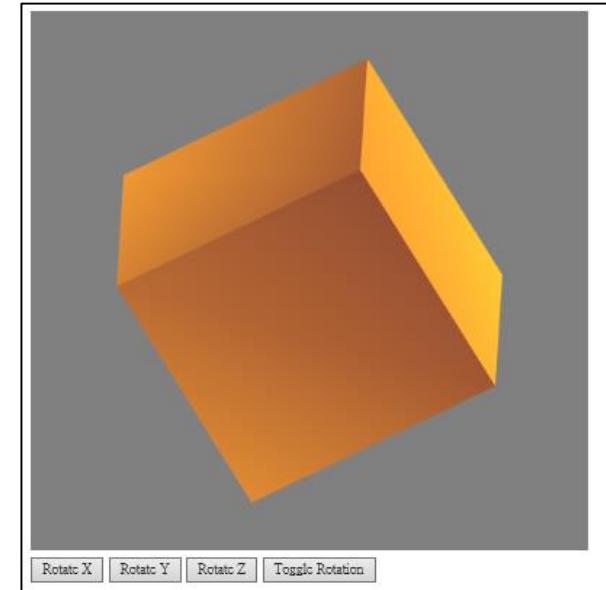
```
gl.uniform1i(gl.getUniformLocation(program, "i"), 0);
gl.clear( gl.COLOR_BUFFER_BIT );
gl.uniform3fv(thetaLoc, theta);
gl.drawArrays(gl.TRIANGLES, 0, 36);

}); // canvas.addEventListener("mousedown", function() {}

render();
} // end of window.onload
```

0: normal rendering

A cube has 6 faces and each face has 2 triangles.
Total vertices = 6 faces x 2 triangles/face x 3 vertices/triangle
= 36 vertices



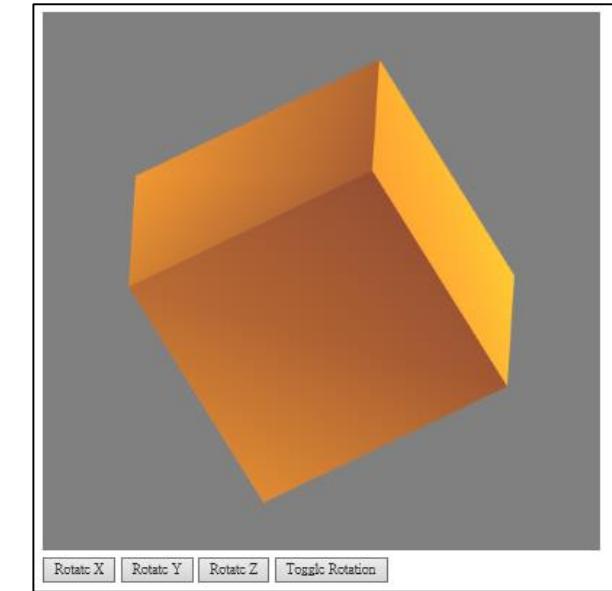
pickCube2.js (15/15)

```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    if(flag) theta[axis] += 2.0;
    modelView = mat4();
    modelView = mult(modelView, rotate(theta[xAxis], [1, 0, 0] ));
    modelView = mult(modelView, rotate(theta[yAxis], [0, 1, 0] ));
    modelView = mult(modelView, rotate(theta[zAxis], [0, 0, 1] ));

    gl.uniformMatrix4fv( gl.getUniformLocation(program, "modelViewMatrix"), false, flatten(modelView) );

    gl.uniform1i(gl.getUniformLocation(program, "i"), 0);
    gl.drawArrays( gl.TRIANGLES, 0, 36 );
}

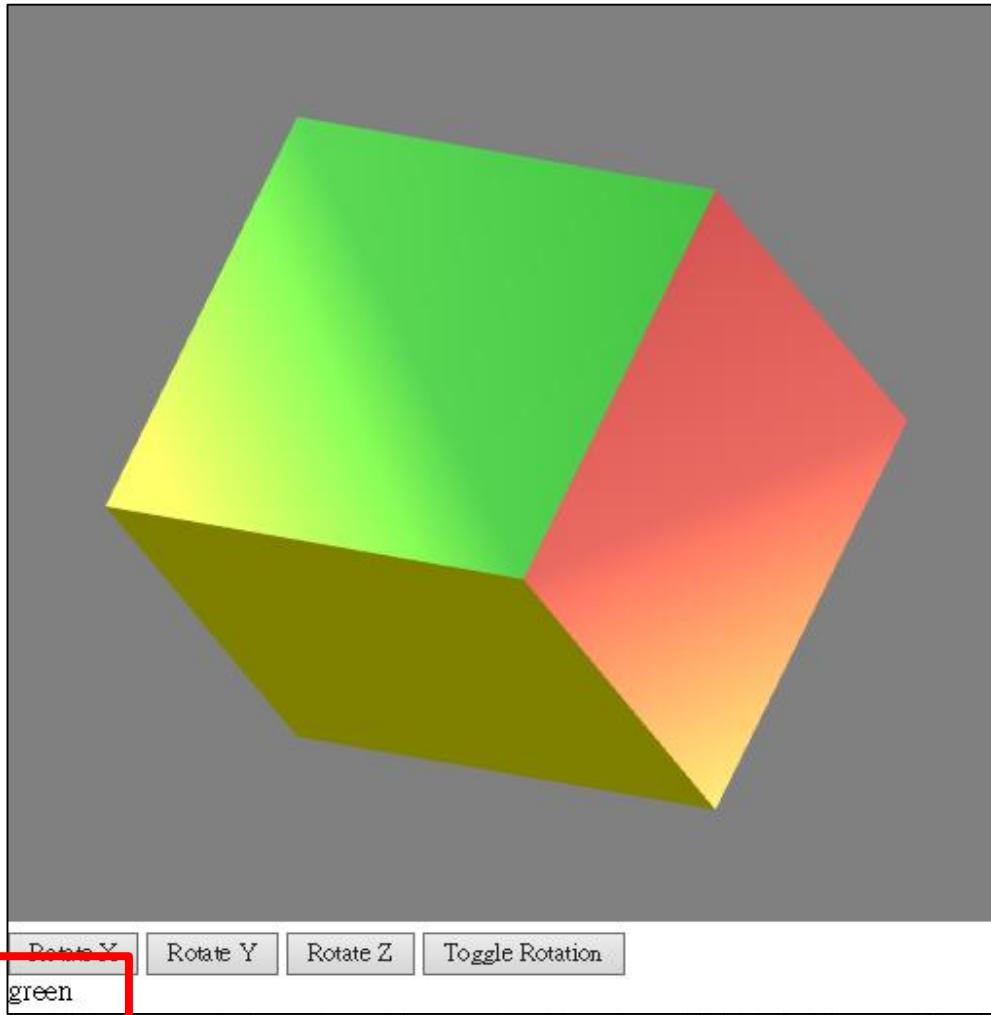
requestAnimationFrame(render);
}
```



0: normal rendering

A cube has 6 faces and each face has 2 triangles.
Total vertices = 6 faces x 2 triangles/face x 3 vertices/triangle
= 36 vertices

Sample Programs: pickCube3.html, pickCube3.js



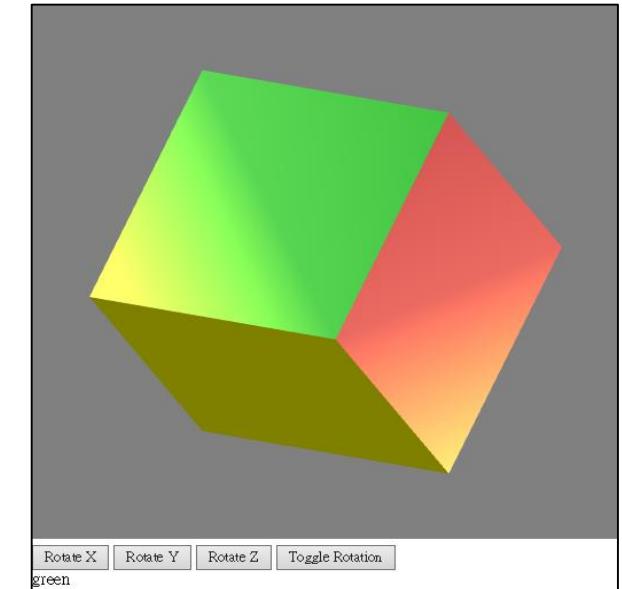
Changes the material properties so each face has a color but lighting still causes varying shades across each face. Name of face color is displayed in window instead of on console

pickCube3.html (1/6)

```
<!DOCTYPE html>
<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="pickCube3.js"></script>

<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
attribute vec3 vNormal;
attribute vec4 vColor;
varying vec4 fColor;

uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float shininess;
```



pickCube3.html (2/6)

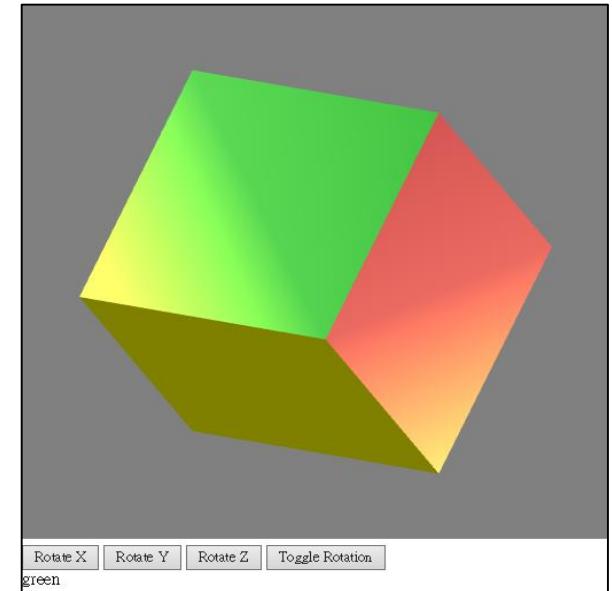
```
void main()
{
    vec3 pos = -(modelViewMatrix * vPosition).xyz;
    vec3 light = lightPosition.xyz;
    vec3 L = normalize( light - pos );

    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E );

    vec4 NN = vec4(vNormal,0);

    // Transform vertex normal into eye coordinates

    vec3 N = normalize( (modelViewMatrix*NN).xyz);
```



pickCube3.html (3/6)

```
// Compute terms in the illumination equation
//vec4 ambient = ambientProduct;
vec4 ambient = 0.5*vColor;

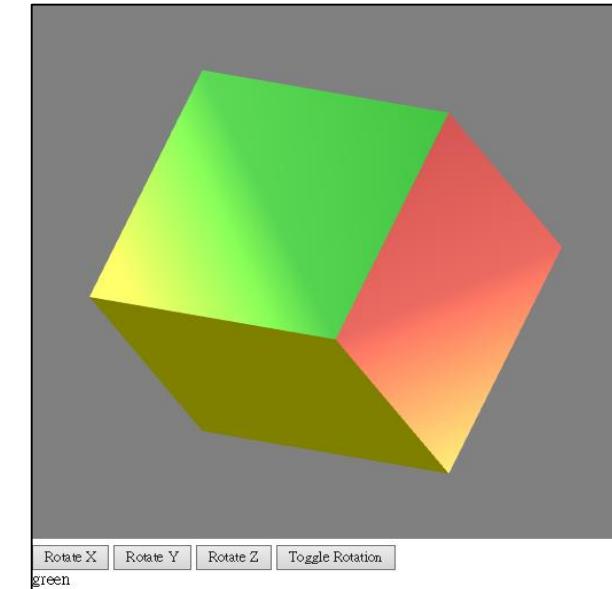
float Kd = max( dot(L, N), 0.0 );
vec4 diffuse = Kd*diffuseProduct;

float Ks = pow( max(dot(N, H), 0.0), shininess );
vec4 specular = Ks * specularProduct;

if( dot(L, N) < 0.0 ) { specular = vec4(0.0, 0.0, 0.0, 1.0); }

gl_Position = projectionMatrix * modelViewMatrix * vPosition;
fColor = ambient + diffuse +specular;

fColor.a = 1.0;
}
```



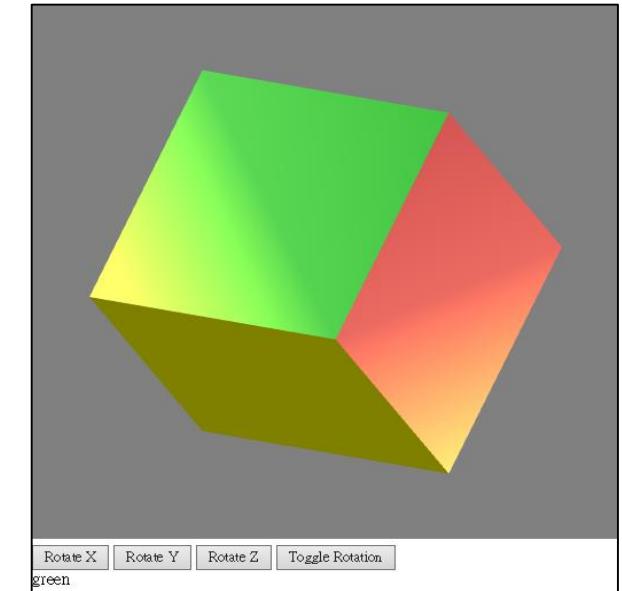
pickCube3.html (4/6)

```
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

uniform int i;

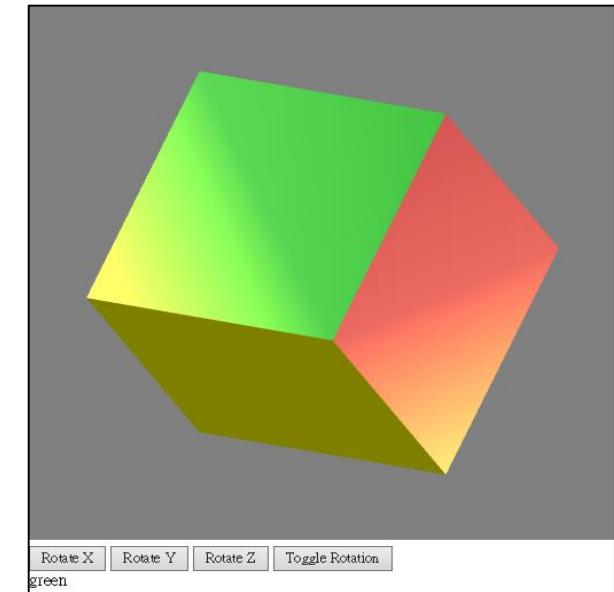
varying vec4 fColor;

void main()
{
    vec4 c[7];
    c[0] = fColor; ← normal rendering
    c[1] = vec4(1.0, 0.0, 0.0, 1.0);
    c[2] = vec4(0.0, 1.0, 0.0, 1.0);
    c[3] = vec4(0.0, 0.0, 1.0, 1.0);
    c[4] = vec4(1.0, 1.0, 0.0, 1.0); ← 6 colors for 6 faces of a cube: render to texture
    c[5] = vec4(0.0, 1.0, 1.0, 1.0);
    c[6] = vec4(1.0, 0.0, 1.0, 1.0);
```



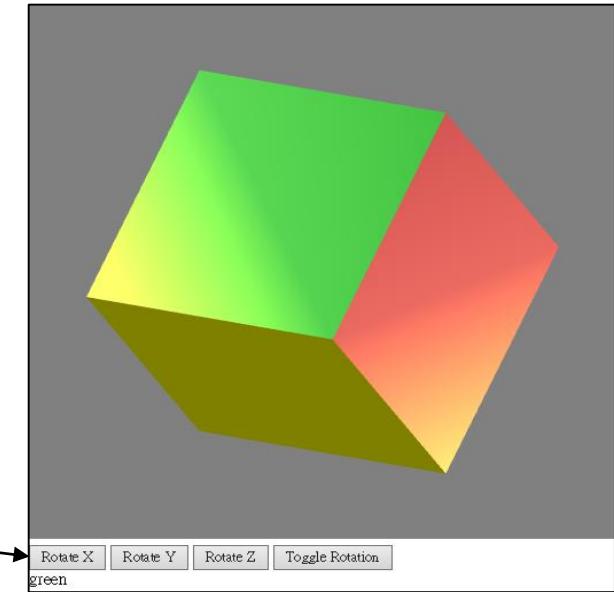
pickCube3.html (5/6)

```
if (i==0) gl_FragColor = c[0]; ← normal rendering  
else if (i==1) gl_FragColor = c[1];  
else if (i==2) gl_FragColor = c[2];  
else if (i==3) gl_FragColor = c[3]; ← i (>0): render to texture  
else if (i==4) gl_FragColor = c[4]; with base colors  
else if (i==5) gl_FragColor = c[5];  
else if (i==6) gl_FragColor = c[6];  
}  
</script>
```



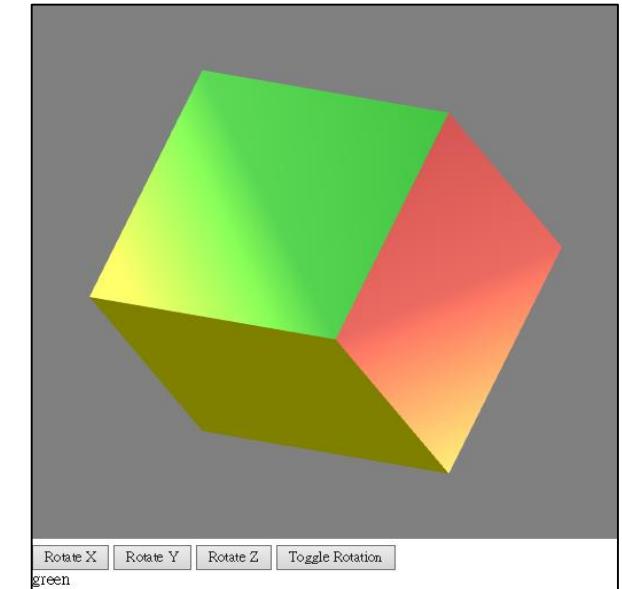
pickCube3.html (6/6)

```
<body>
<div>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</div>
<div>
<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
<button id = "ButtonT">Toggle Rotation</button>
</div>
<div id = "test">
face
</div>
</body>
</html>
```



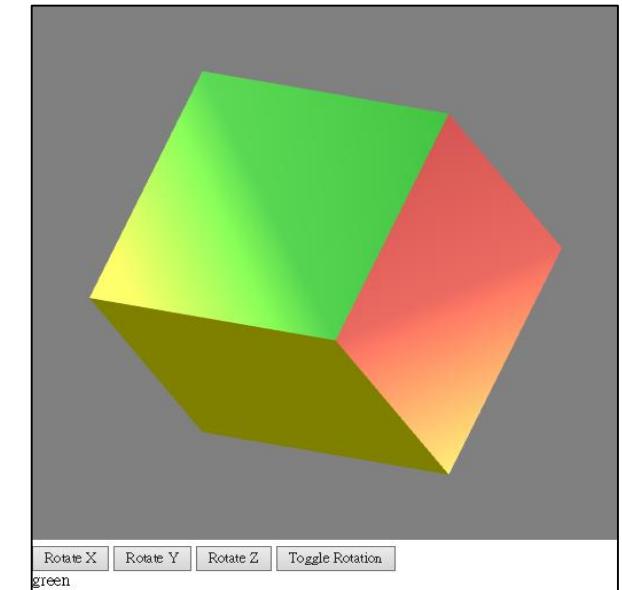
pickCube3.js (1/18)

```
var elt;  
  
var canvas;  
var gl;  
  
var program;  
  
var NumVertices = 36;  
  
var pointsArray = [];  
var normalsArray = [];  
var colorsArray = [];  
  
var framebuffer;  
  
var flag = true;  
  
var color = new Uint8Array(4);
```



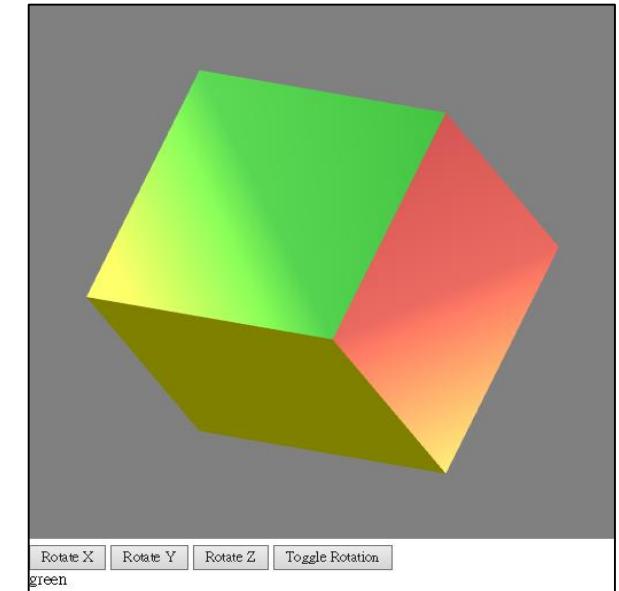
pickCube3.js (2/18)

```
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 ),  
];
```



pickCube3.js (3/18)

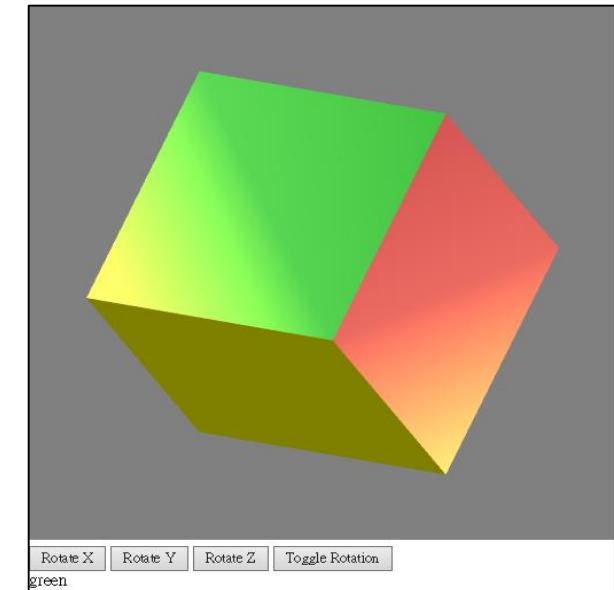
```
var vertexColors = [  
    vec4( 0.0, 0.0, 0.0, 1.0 ), // black  
    vec4( 1.0, 0.0, 0.0, 1.0 ), // red  
    vec4( 1.0, 1.0, 0.0, 1.0 ), // yellow  
    vec4( 0.0, 1.0, 0.0, 1.0 ), // green  
    vec4( 0.0, 0.0, 1.0, 1.0 ), // blue  
    vec4( 1.0, 0.0, 1.0, 1.0 ), // magenta  
    vec4( 0.0, 1.0, 1.0, 1.0 ), // cyan  
    vec4( 1.0, 1.0, 1.0, 1.0 ), // white  
];
```



pickCube3.js (4/18)

```
var lightPosition = vec4(1.0, 1.0, 1.0, 0.0);
var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0 );

//var materialAmbient = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialAmbient = vec4( 1.0, 1.0, 1.0, 1.0 );
//var materialDiffuse = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialDiffuse = vec4( 0.5, 0.5, 0.5, 1.0 );
var materialSpecular = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 10.0;
```



pickCube3.js (5/18)

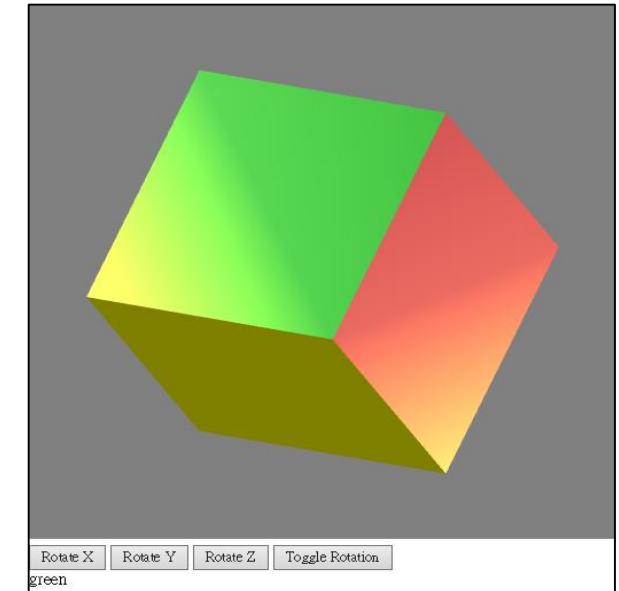
```
var ctm;
var ambientColor, diffuseColor, specularColor;
var modelView, projection;
var viewerPos;
var program;

var xAxis = 0;
var yAxis = 1;
var zAxis = 2;
var axis = xAxis;

var theta = [45.0, 45.0, 45.0];

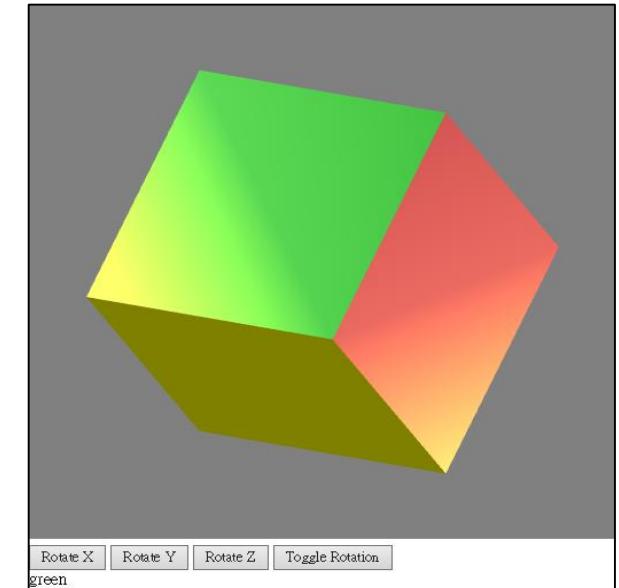
var thetaLoc;

var Index = 0;
```



pickCube3.js (6/18)

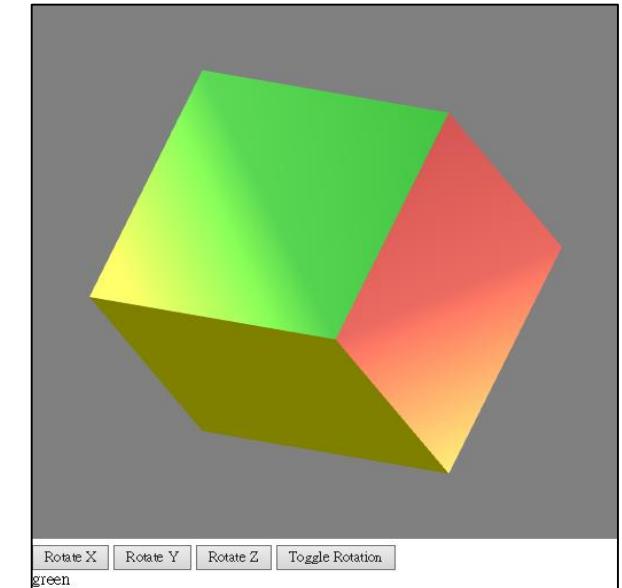
```
function quad(a, b, c, d) {  
  
    var t1 = subtract(vertices[b], vertices[a]);  
    var t2 = subtract(vertices[c], vertices[b]);  
    var normal = cross(t1, t2);  
    var normal = vec3(normal);  
    normal = normalize(normal);  
  
    pointsArray.push(vertices[a]);  
    normalsArray.push(normal);  
    colorsArray.push(vertexColors[a]);  
    pointsArray.push(vertices[b]);  
    normalsArray.push(normal);  
    colorsArray.push(vertexColors[a]);  
    pointsArray.push(vertices[c]);  
    normalsArray.push(normal);  
    colorsArray.push(vertexColors[a]);
```



pickCube3.js (7/18)

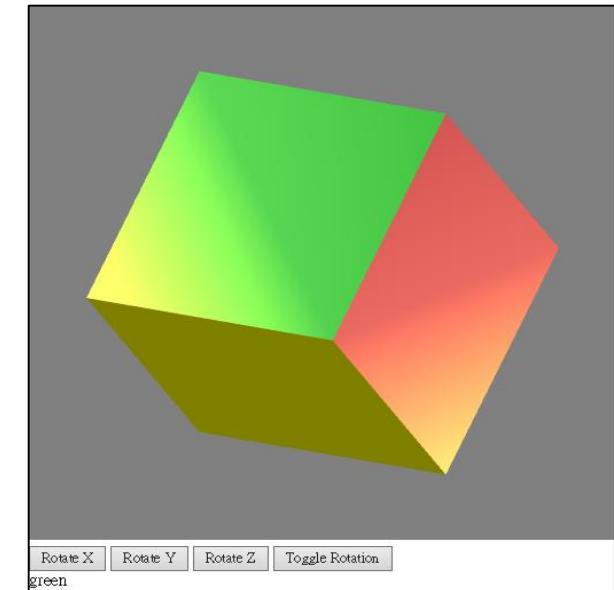
```
pointsArray.push(vertices[a]);
normalsArray.push(normal);
colorsArray.push(vertexColors[a]);
pointsArray.push(vertices[c]);
normalsArray.push(normal);
colorsArray.push(vertexColors[a]);
pointsArray.push(vertices[d]);
normalsArray.push(normal);
colorsArray.push(vertexColors[a]);
}

// end of quad(a, b, c, d)
```



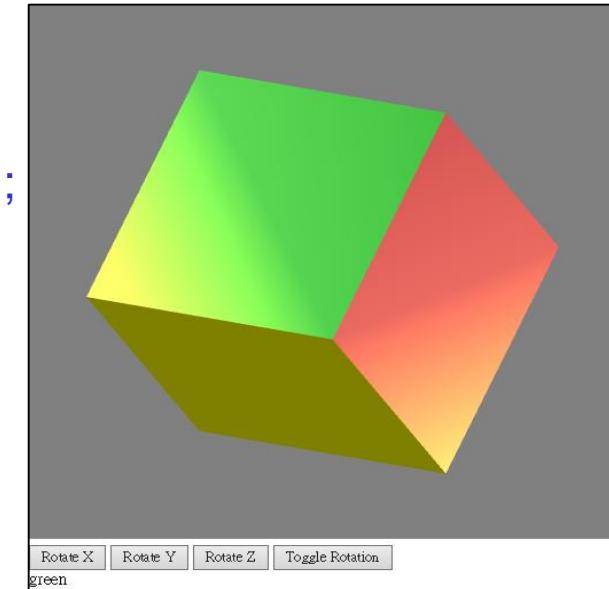
pickCube3.js (8/18)

```
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 );
}
```



pickCube3.js (9/18)

```
window.onload = function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    var ctx = canvas.getContext("experimental-webgl", {preserveDrawingBuffer: true});  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    elt = document.getElementById("test");  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 0.5, 0.5, 0.5, 1.0 );  
  
    gl.enable(gl.CULL_FACE);
```



pickCube3.js (10/18)

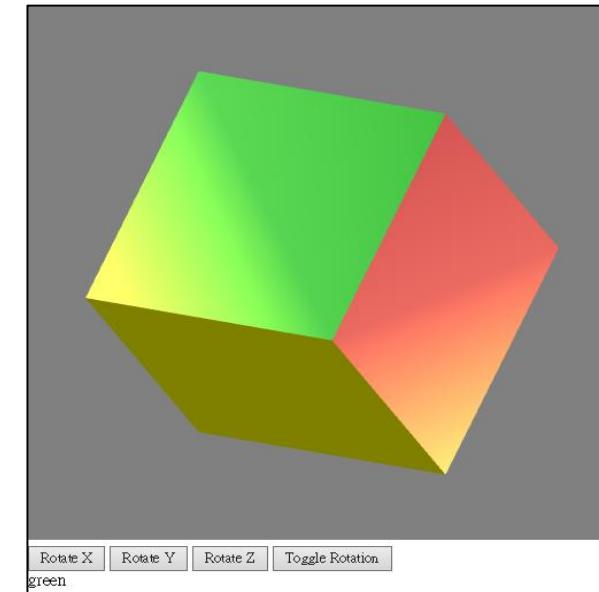
```
var texture = gl.createTexture();
gl.bindTexture( gl.TEXTURE_2D, texture );
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 512, 512, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);
gl.generateMipmap(gl.TEXTURE_2D);

// Allocate a frame buffer object

framebuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, framebuffer);

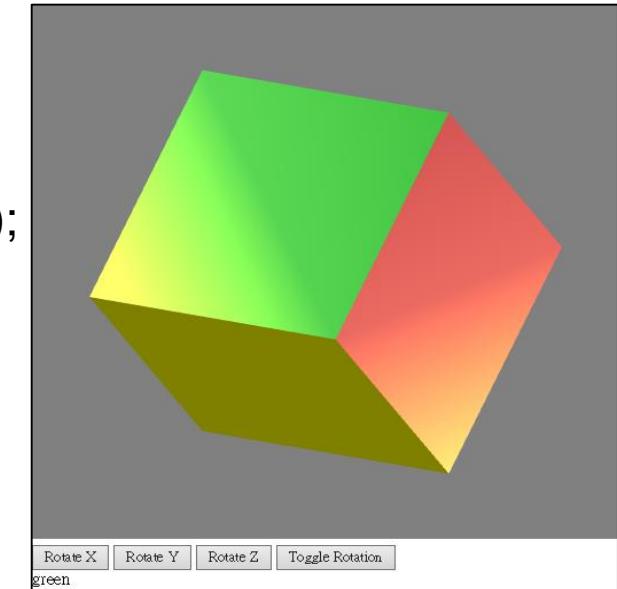
// Attach color buffer

gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);
```



pickCube3.js (11/18)

```
// check for completeness  
  
//var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);  
//if(status != gl.FRAMEBUFFER_COMPLETE) alert('Frame Buffer Not Complete');  
  
gl.bindFramebuffer(gl.FRAMEBUFFER, null);  
  
//  
// Load shaders and initialize attribute buffers  
//  
program = initShaders( gl, "vertex-shader", "fragment-shader" );  
gl.useProgram( program );  
  
colorCube();
```



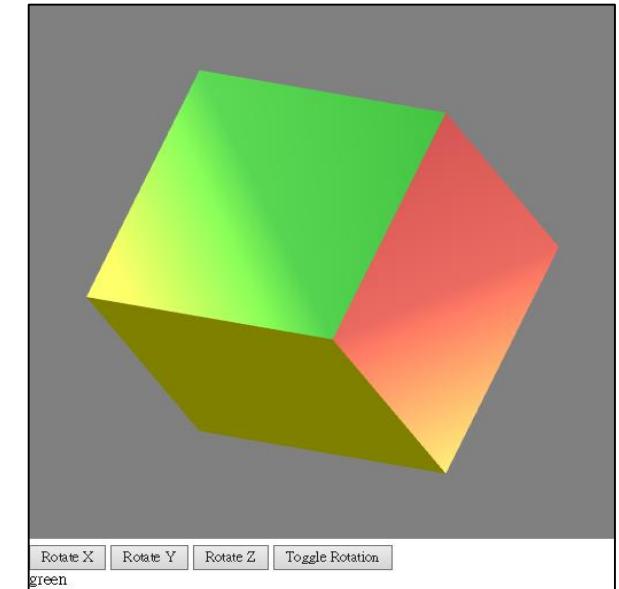
pickCube3.js (12/18)

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

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

var nBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, nBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW );

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



pickCube3.js (13/18)

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

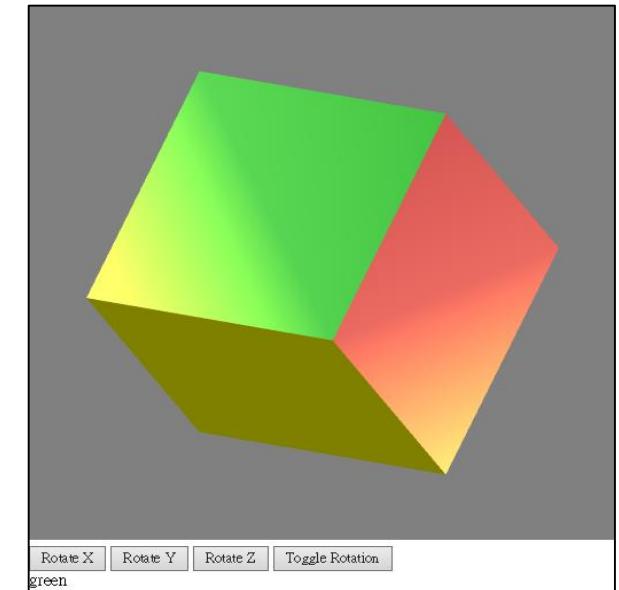
var vPosition = gl.getAttribLocation(program, "vPosition");
gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);

thetaLoc = gl.getUniformLocation(program, "theta");

viewerPos = vec3(0.0, 0.0, -20.0 );

projection = ortho(-1, 1, -1, 1, -100, 100);

ambientProduct = mult(lightAmbient, materialAmbient);
diffuseProduct = mult(lightDiffuse, materialDiffuse);
specularProduct = mult(lightSpecular, materialSpecular);
```



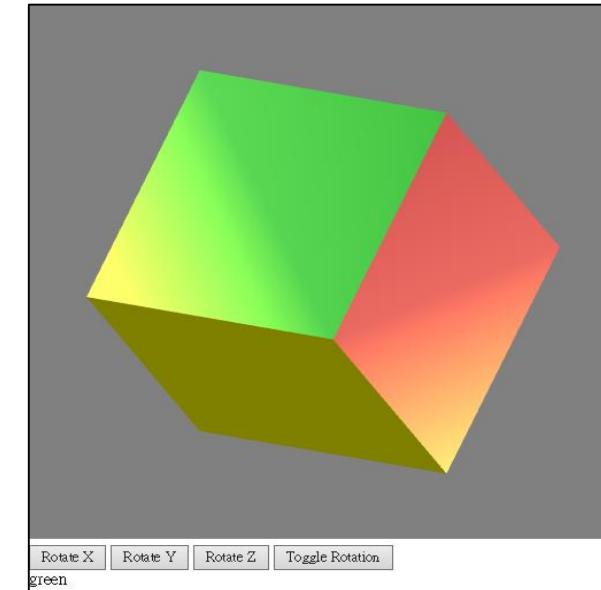
pickCube3.js (14/18)

```
document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};
document.getElementById("ButtonT").onclick = function() {flag = !flag};
```

```
gl.uniform4fv(gl.getUniformLocation(program, "ambientProduct"), flatten(ambientProduct));
gl.uniform4fv(gl.getUniformLocation(program, "diffuseProduct"), flatten(diffuseProduct) );
gl.uniform4fv(gl.getUniformLocation(program, "specularProduct"), flatten(specularProduct) );
gl.uniform4fv(gl.getUniformLocation(program, "lightPosition"), flatten(lightPosition) );

gl.uniform1f(gl.getUniformLocation(program, "shininess"), materialShininess);

gl.uniformMatrix4fv( gl.getUniformLocation(program, "projectionMatrix"), false, flatten(projection));
```

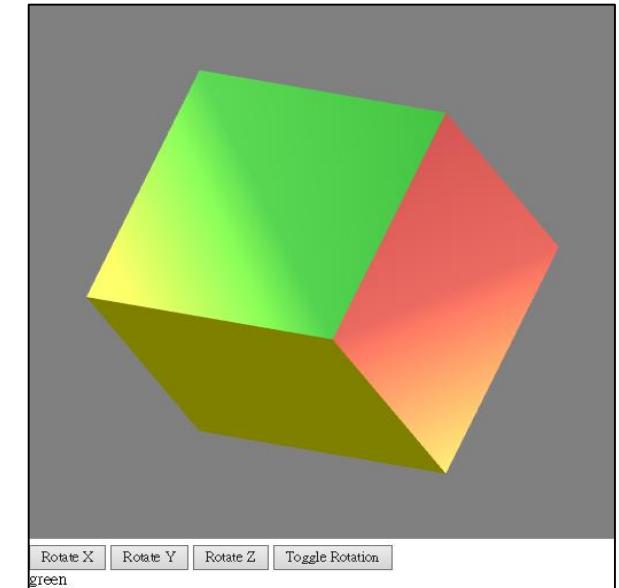


pickCube3.js (15/18)

```
canvas.addEventListener("mousedown", function() {  
  
    gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);  
    gl.clear( gl.COLOR_BUFFER_BIT );  
    gl.uniform3fv(thetaLoc, theta);  
    for(var i=0; i<6; i++) {  
        gl.uniform1i(gl.getUniformLocation(program, "i"), i+1);  
        gl.drawArrays( gl.TRIANGLES, 6*i, 6 );  
    }  
});
```

i+1 (>0): render to texture

Draw face i (2 triangles with 6 vertices).
Each face has different color. (color[i+1])



pickCube3.js (16/18)

```
var x = event.clientX;  
var y = canvas.height - event.clientY;
```

Get mouse position

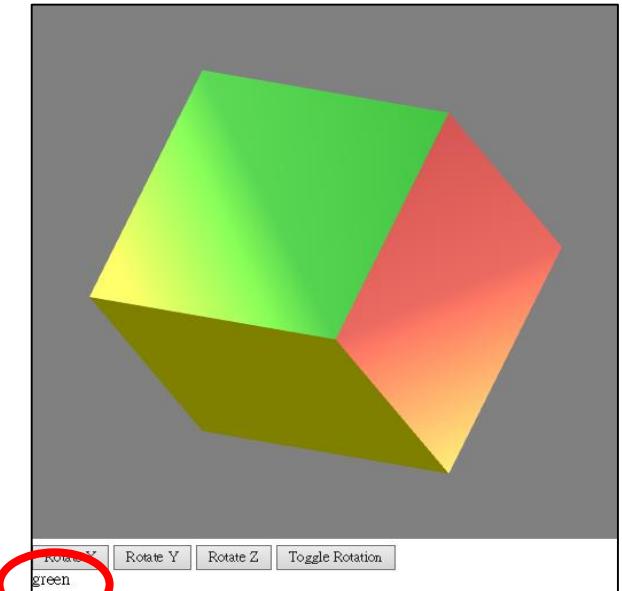
```
gl.readPixels(x, y, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, color);
```

```
if(color[0]==255)  
    if(color[1]==255) elt.innerHTML = "<div> cyan </div>";  
    else if(color[2]==255) elt.innerHTML = "<div> magenta </div>";  
    else elt.innerHTML = "<div> red </div>";  
else if(color[1]==255)  
    if(color[2]==255) elt.innerHTML = "<div> blue </div>";  
    else elt.innerHTML = "<div> yellow </div>";  
else if(color[2]==255) elt.innerHTML = "<div> green </div>";  
    else elt.innerHTML = "<div> background </div>";
```

Get color with mouse location and output it

```
gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

Find which face was read and output it



pickCube3.js (17/18)

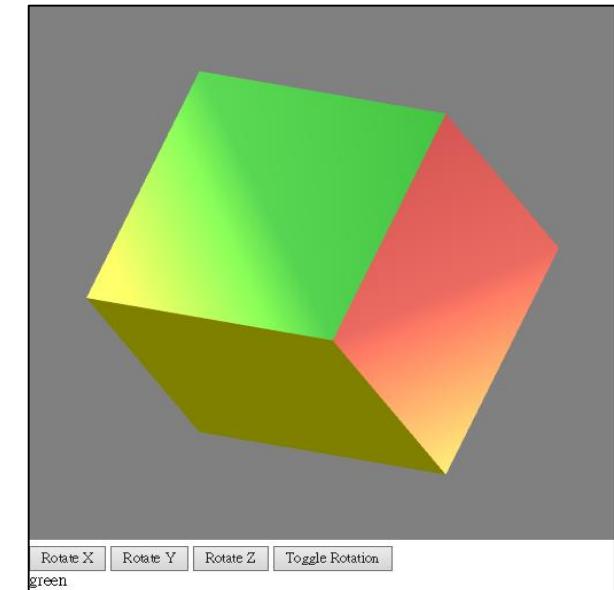
```
gl.uniform1i(gl.getUniformLocation(program, "i"), 0);
gl.clear( gl.COLOR_BUFFER_BIT );
gl.uniform3fv(thetaLoc, theta);
gl.drawArrays(gl.TRIANGLES, 0, 36);

}); // end of canvas.addEventListener("mousedown", function(){}

render();

} // end of window.onload
```

A cube has 6 faces and each face has 2 triangles.
Total vertices = 6 faces x 2 triangles/face x 3 vertices/triangle
= 36 vertices



pickCube3.js (18/18)

```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    if(flag) theta[axis] += 2.0;
    modelView = mat4();
    modelView = mult(modelView, rotate(theta[xAxis], [1, 0, 0]));
    modelView = mult(modelView, rotate(theta[yAxis], [0, 1, 0]));
    modelView = mult(modelView, rotate(theta[zAxis], [0, 0, 1]));

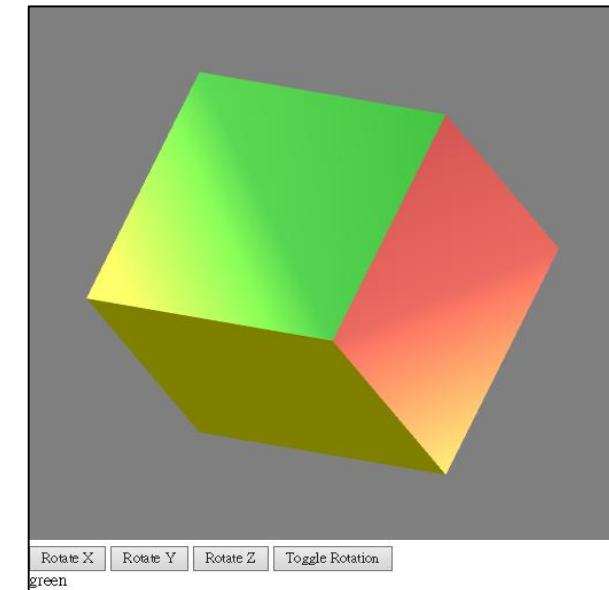
    gl.uniformMatrix4fv( gl.getUniformLocation(program, "modelViewMatrix"), false, flatten(modelView) );

    gl.uniform1i(gl.getUniformLocation(program, "i"), 0);
    gl.drawArrays( gl.TRIANGLES, 0, 36 );
}

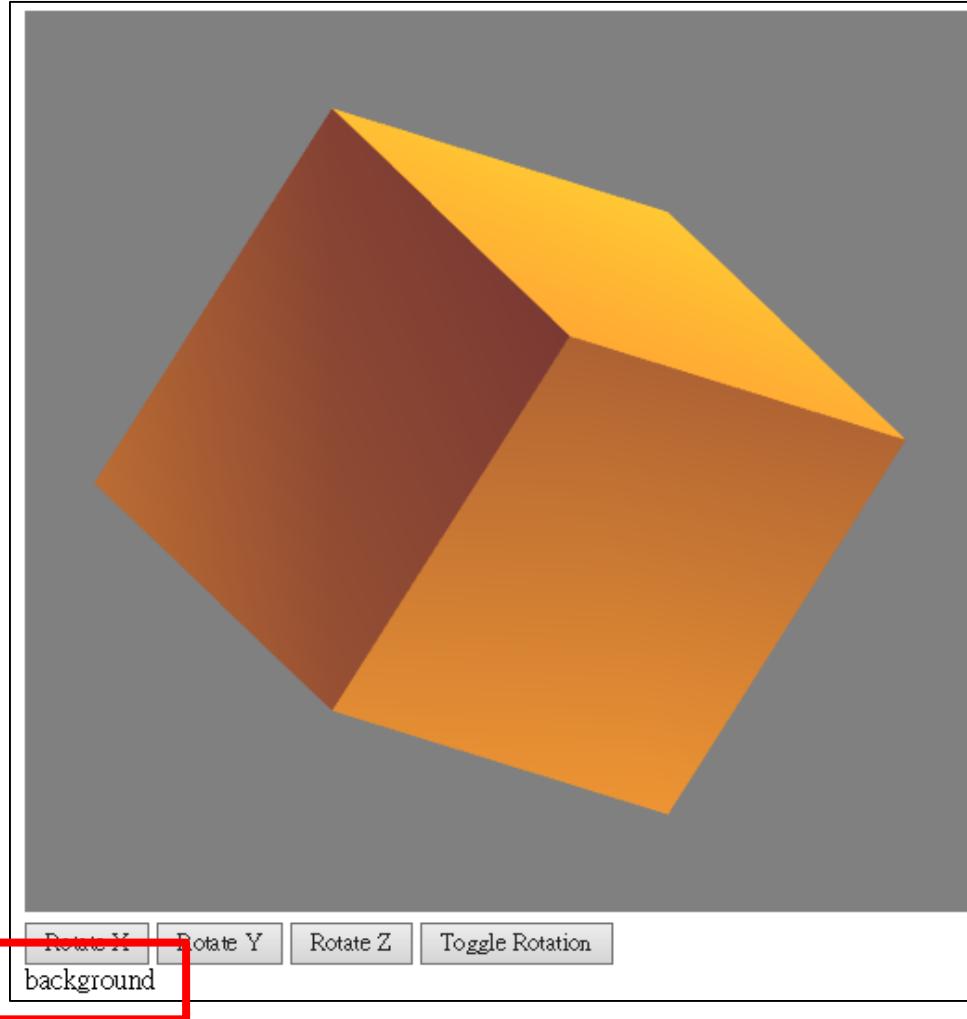
requestAnimFrame(render);
}
```

0: normal rendering

A cube has 6 faces and each face has 2 triangles.
Total vertices = 6 faces x 2 triangles/face x 3 vertices/triangle
= 36 vertices



Sample Programs: pickCube4.html, pickCube4.js



Similar to pickCube2 but displays name of picked face in window

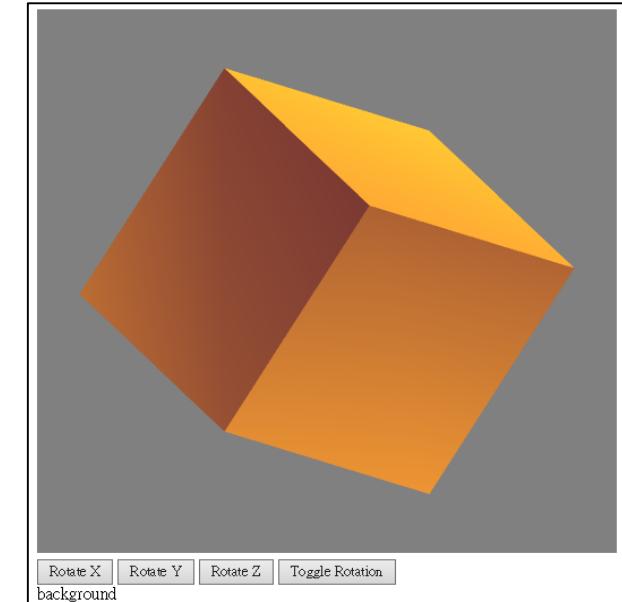
pickCube4.html (1/6)

```
<!DOCTYPE html>
<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="pickCube4.js"></script>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec3 vNormal;
varying vec4 fColor;

uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float shininess;
```



pickCube4.html (2/6)

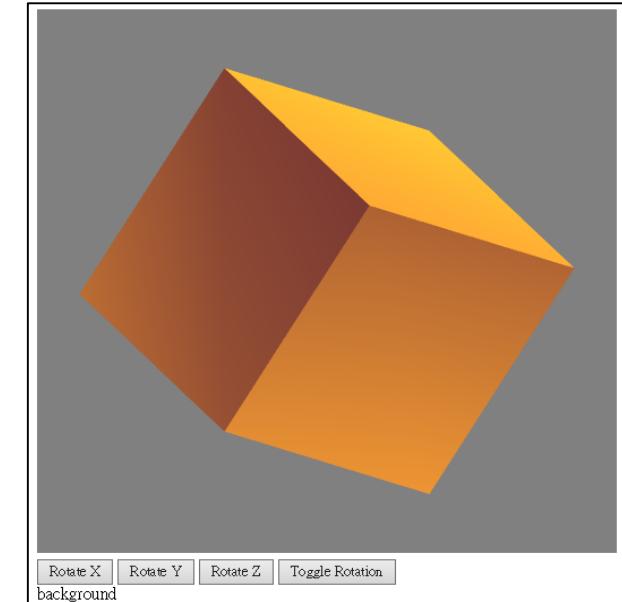
```
void main()
{
    vec3 pos = -(modelViewMatrix * vPosition).xyz;
    vec3 light = lightPosition.xyz;
    vec3 L = normalize( light - pos );

    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E );

    vec4 NN = vec4(vNormal,0);

// Transform vertex normal into eye coordinates

    vec3 N = normalize( (modelViewMatrix*NN).xyz);
```



pickCube4.html (3/6)

```
// Compute terms in the illumination equation
vec4 ambient = ambientProduct;

float Kd = max( dot(L, N), 0.0 );
vec4 diffuse = Kd*diffuseProduct;

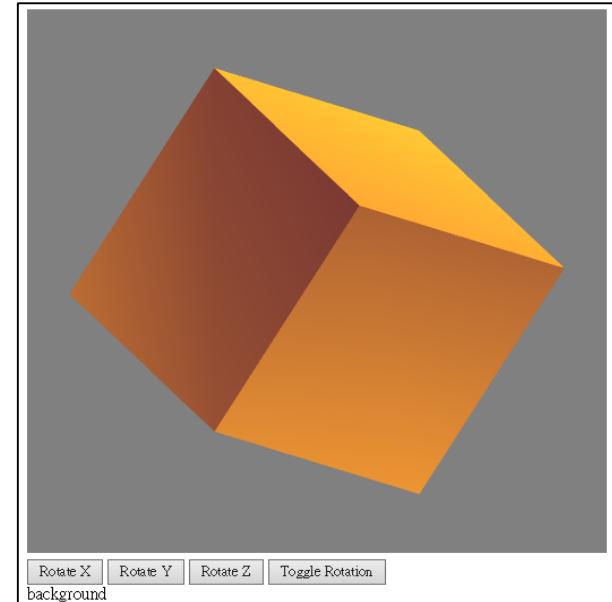
float Ks = pow( max(dot(N, H), 0.0), shininess );
vec4 specular = Ks * specularProduct;

if( dot(L, N) < 0.0 ) { specular = vec4(0.0, 0.0, 0.0, 1.0); }

gl_Position = projectionMatrix * modelViewMatrix * vPosition;
fColor = ambient + diffuse + specular;

fColor.a = 1.0;
}

</script>
```



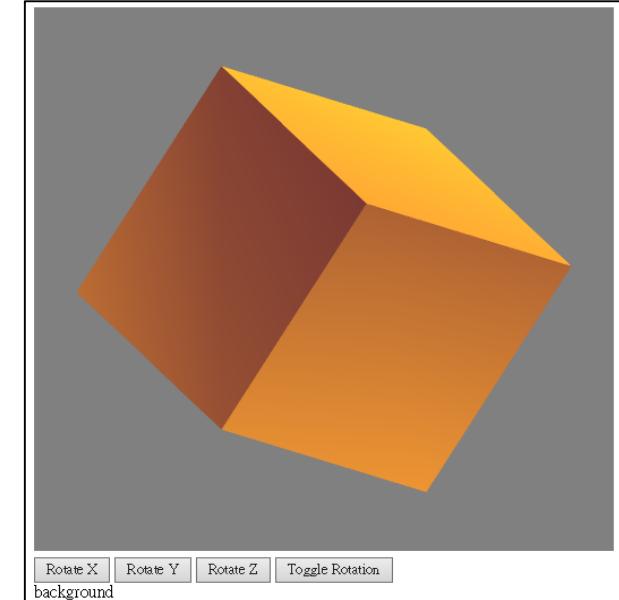
pickCube4.html (4/6)

```
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

uniform int i;

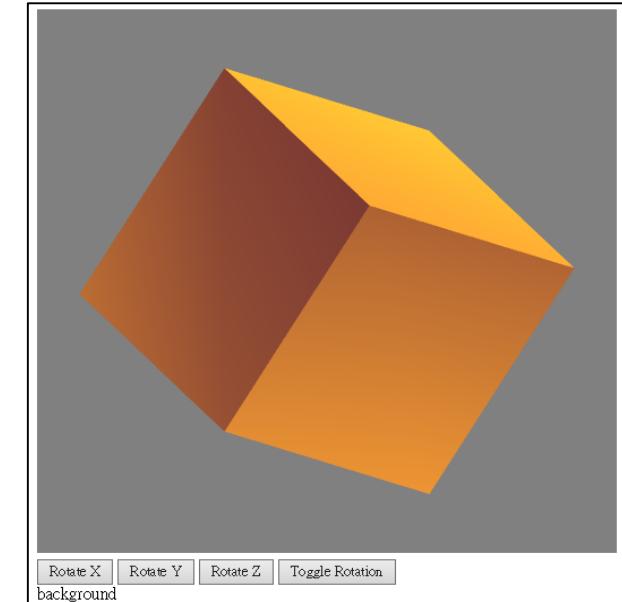
varying vec4 fColor;

void
main()
{
    vec4 c[7];
    c[0] = fColor; ← normal rendering
    c[1] = vec4(1.0, 0.0, 0.0, 1.0);
    c[2] = vec4(0.0, 1.0, 0.0, 1.0);
    c[3] = vec4(0.0, 0.0, 1.0, 1.0);
    c[4] = vec4(1.0, 1.0, 0.0, 1.0); ← 6 colors for 6 faces of a cube: render to texture
    c[5] = vec4(0.0, 1.0, 1.0, 1.0);
    c[6] = vec4(1.0, 0.0, 1.0, 1.0);
```



pickCube4.html (5/6)

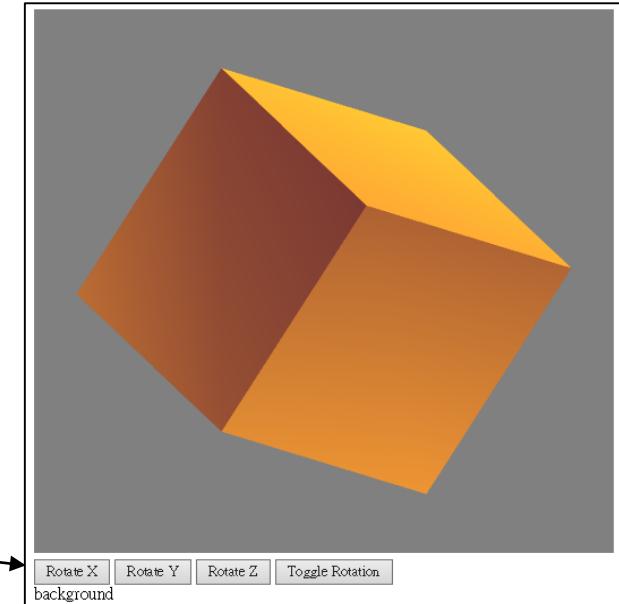
```
if (i==0) gl_FragColor = c[0]; ← normal rendering  
else if (i==1) gl_FragColor = c[1];  
else if (i==2) gl_FragColor = c[2];  
else if (i==3) gl_FragColor = c[3]; ← i (>0): render to texture  
else if (i==4) gl_FragColor = c[4]; with base colors  
else if (i==5) gl_FragColor = c[5];  
else if (i==6) gl_FragColor = c[6];  
}  
</script>
```



pickCube4.html (6/6)

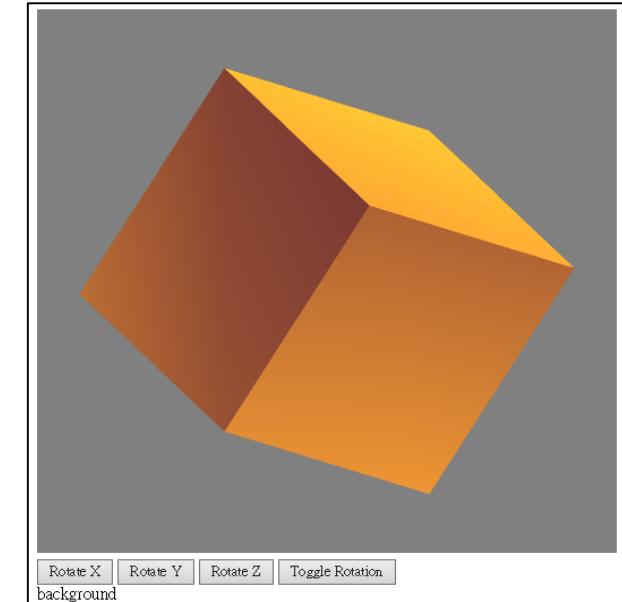
```
<body>
<div>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</div>
<div>
<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
<button id = "ButtonT">Toggle Rotation</button>
</div>
<div id = "test">
face
</div>

</body>
</html>
```



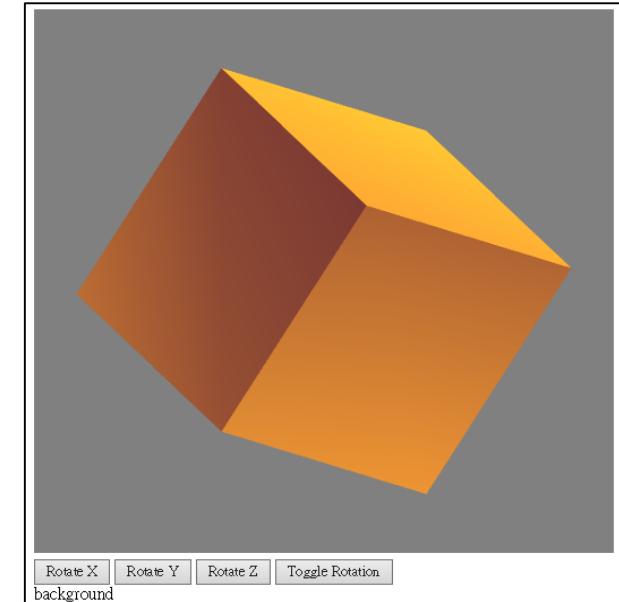
pickCube4.js (1/16)

```
var elt;  
  
var canvas;  
var gl;  
  
var program;  
  
var NumVertices = 36;  
  
var pointsArray = [];  
var normalsArray = [];  
  
var framebuffer;  
  
var flag = false;  
  
var color = new Uint8Array(4);
```



pickCube4.js (2/16)

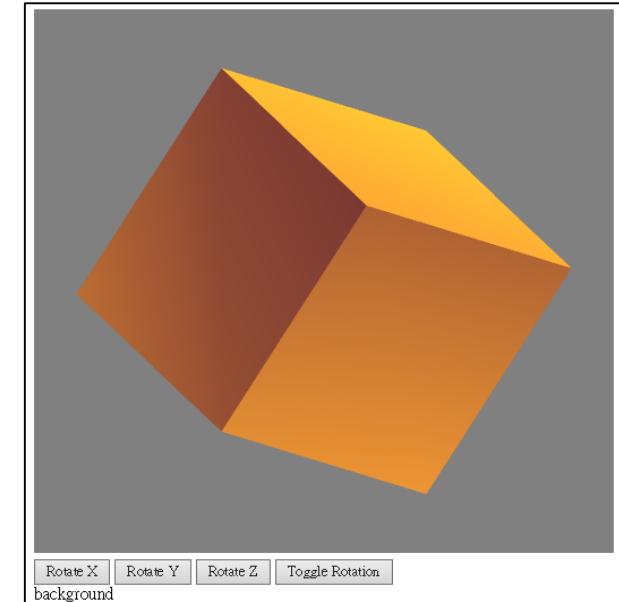
```
var vertices = [  
    vec4( -0.5, -0.5,  0.5, 1.0 ),  
    vec4( -0.5,  0.5,  0.5, 1.0 ),  
    vec4(  0.5,  0.5,  0.5, 1.0 ),  
    vec4(  0.5, -0.5,  0.5, 1.0 ),  
    vec4( -0.5, -0.5, -0.5, 1.0 ),  
    vec4( -0.5,  0.5, -0.5, 1.0 ),  
    vec4(  0.5,  0.5, -0.5, 1.0 ),  
    vec4(  0.5, -0.5,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



pickCube4.js (3/16)

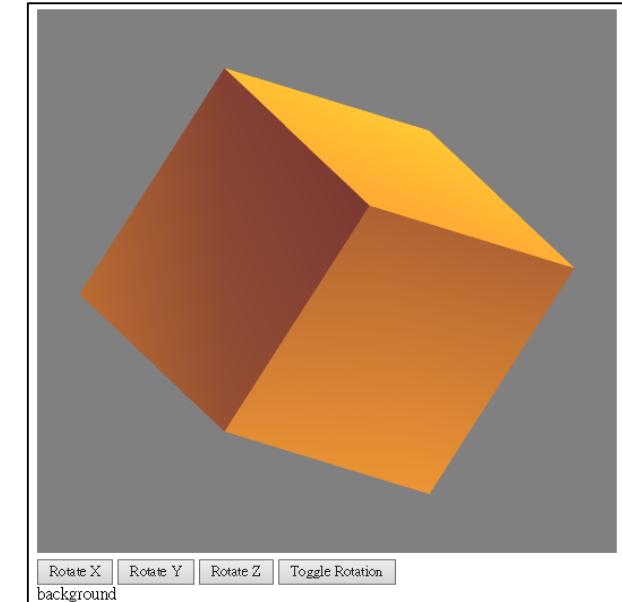
```
var lightPosition = vec4( 1.0, 1.0, 1.0, 0.0 );
var lightAmbient = vec4( 0.2, 0.2, 0.2, 1.0 );
var lightDiffuse = vec4( 1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4( 1.0, 1.0, 1.0, 1.0 );

var materialAmbient = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialSpecular = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;
```



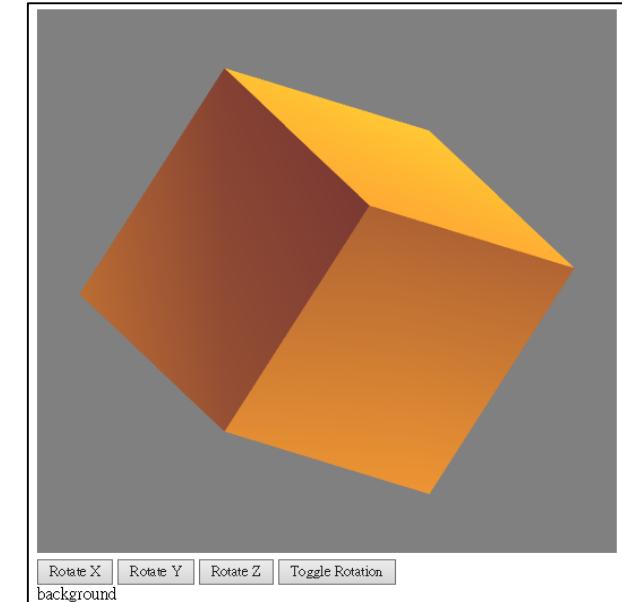
pickCube4.js (4/16)

```
var ctm;  
var ambientColor, diffuseColor, specularColor;  
var modelView, projection;  
var viewerPos;  
var program;  
  
var xAxis = 0;  
var yAxis = 1;  
var zAxis = 2;  
var axis = xAxis;  
  
var theta = [45.0, 45.0, 45.0];  
  
var thetaLoc;  
  
var Index = 0;
```



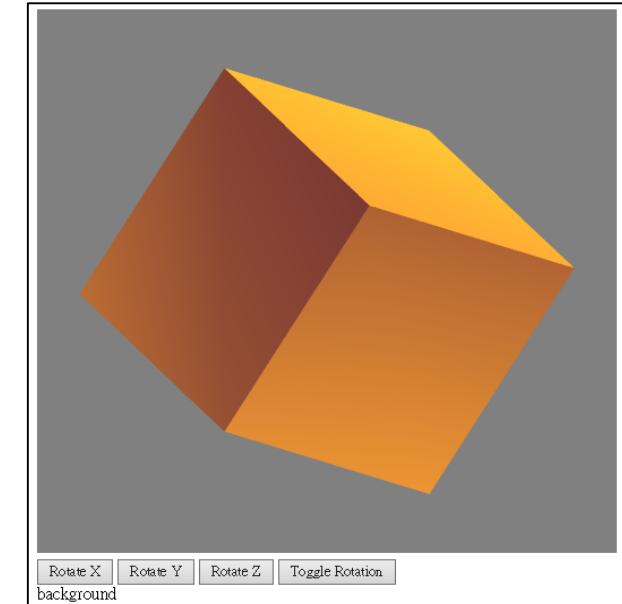
pickCube4.js (5/16)

```
function quad(a, b, c, d) {  
    var t1 = subtract(vertices[b], vertices[a]);  
    var t2 = subtract(vertices[c], vertices[b]);  
    var normal = cross(t1, t2);  
    var normal = vec3(normal);  
    normal = normalize(normal);  
    pointsArray.push(vertices[a]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[b]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[c]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[a]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[c]);  
    normalsArray.push(normal);  
    pointsArray.push(vertices[d]);  
    normalsArray.push(normal);  
}
```



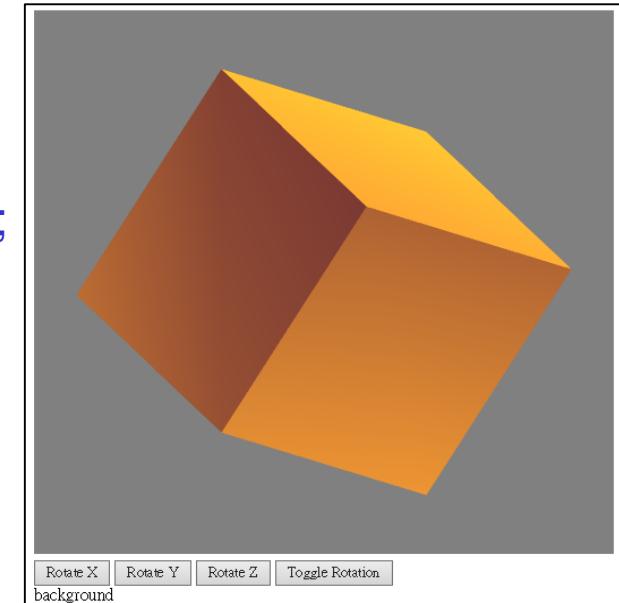
pickCube4.js (6/16)

```
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 );
}
```



pickCube4.js (7/16)

```
window.onload = function init() {  
    canvas = document.getElementById( "gl-canvas" );  
  
    var ctx = canvas.getContext("experimental-webgl", {preserveDrawingBuffer: true});  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    elt = document.getElementById("test");  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 0.5, 0.5, 0.5, 1.0 );  
  
    gl.enable(gl.CULL_FACE);
```



pickCube4.js (8/16)

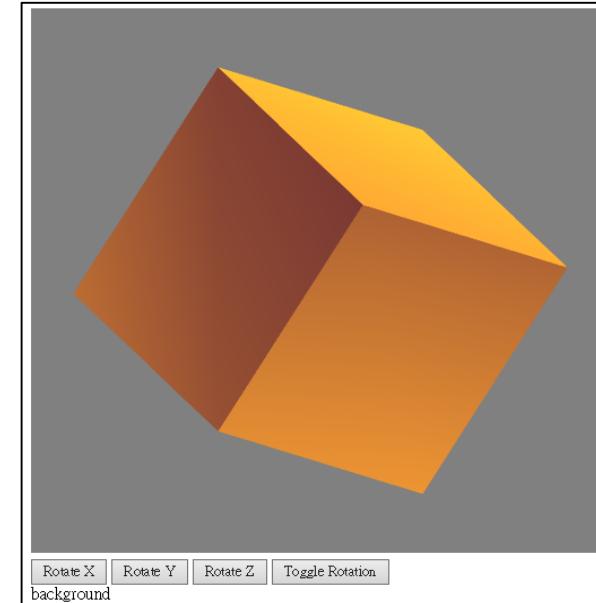
```
var texture = gl.createTexture();
gl.bindTexture( gl.TEXTURE_2D, texture );
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 512, 512, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);
gl.generateMipmap(gl.TEXTURE_2D);

// Allocate a frame buffer object

framebuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, framebuffer);

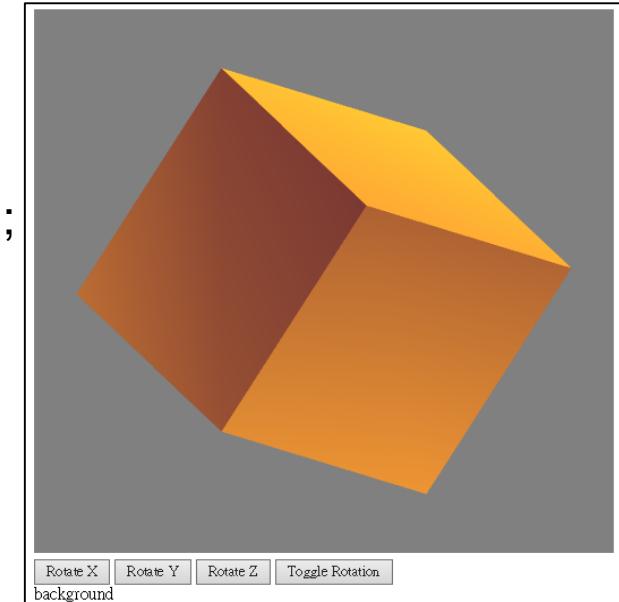
// Attach color buffer

gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);
```



pickCube4.js (9/16)

```
// check for completeness  
  
//var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);  
//if(status != gl.FRAMEBUFFER_COMPLETE) alert('Frame Buffer Not Complete');  
  
gl.bindFramebuffer(gl.FRAMEBUFFER, null);  
  
//  
// Load shaders and initialize attribute buffers  
//  
program = initShaders( gl, "vertex-shader", "fragment-shader" );  
gl.useProgram( program );  
  
colorCube();
```



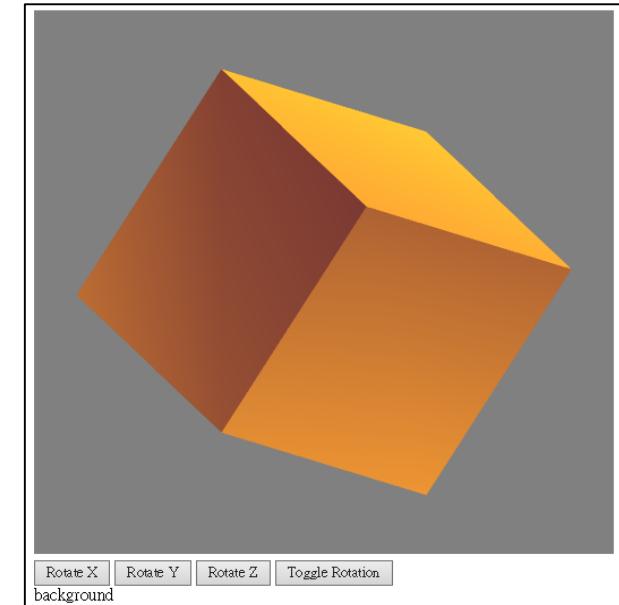
pickCube4.js (10/16)

```
var nBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, nBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW );

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

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

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



pickCube4.js (11/16)

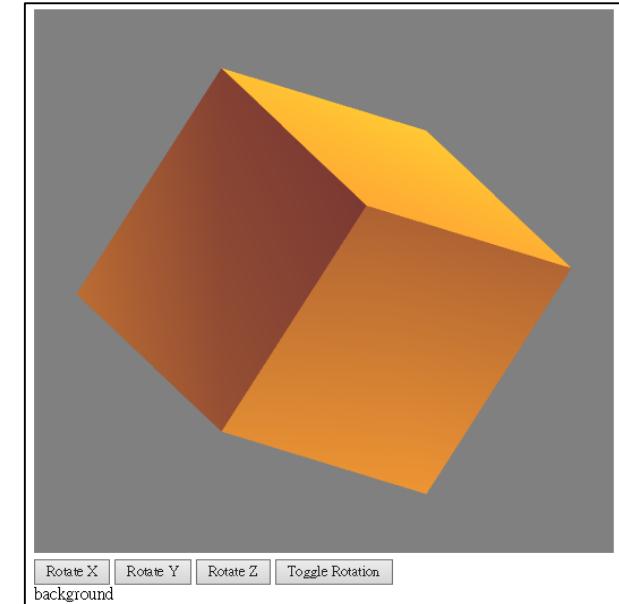
```
thetaLoc = gl.getUniformLocation(program, "theta");

viewerPos = vec3(0.0, 0.0, -20.0 );

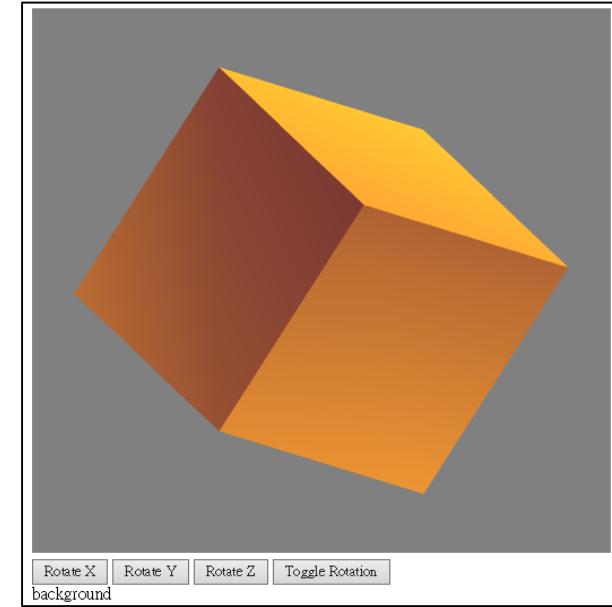
projection = ortho(-1, 1, -1, 1, -100, 100);

ambientProduct = mult(lightAmbient, materialAmbient);
diffuseProduct = mult(lightDiffuse, materialDiffuse);
specularProduct = mult(lightSpecular, materialSpecular);

document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};
document.getElementById("ButtonT").onclick = function() {flag = !flag};
```



pickCube4.js (12/16)



```
gl.uniform4fv(gl.getUniformLocation(program, "ambientProduct"), flatten(ambientProduct) );
gl.uniform4fv(gl.getUniformLocation(program, "diffuseProduct"), flatten(diffuseProduct) );
gl.uniform4fv(gl.getUniformLocation(program, "specularProduct"), flatten(specularProduct) );
gl.uniform4fv(gl.getUniformLocation(program, "lightPosition"), flatten(lightPosition) );

gl.uniform1f(gl.getUniformLocation(program, "shininess"), materialShininess);

gl.uniformMatrix4fv( gl.getUniformLocation(program, "projectionMatrix"), false, flatten(projection));
```

pickCube4.js (13/16)

```
canvas.addEventListener("mousedown", function() {
```

```
    gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
```

```
    gl.clear( gl.COLOR_BUFFER_BIT);
```

```
    gl.uniform3fv(thetaLoc, theta);
```

```
    for(var i=0; i<6; i++) {
```

```
        gl.uniform1i(gl.getUniformLocation(program, "i"), i+1);
```

```
        gl.drawArrays( gl.TRIANGLES, 6*i, 6 );
```

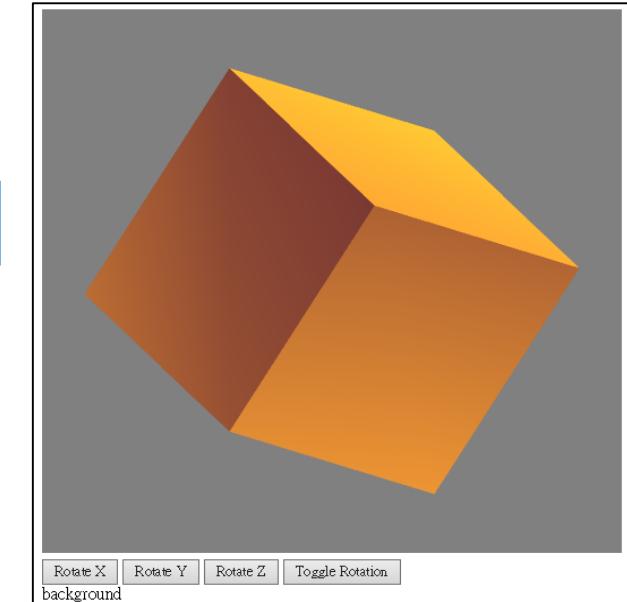
```
}
```

```
    var x = event.clientX;
```

```
    var y = canvas.height -event.clientY;
```

```
    gl.readPixels(x, y, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, color);
```

i+1 (>0): render to texture



Draw face i (2 triangles with **6** vertices).
Each face has different color. ($\text{color}[i+1]$)

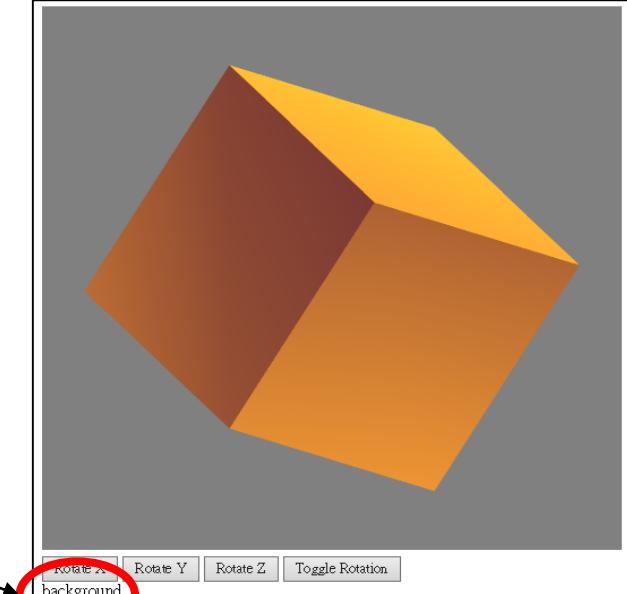
Get mouse position

Get color with mouse location and output it

pickCube4.js (14/16)

```
if (color[0]==255)
    if (color[1]==255) elt.innerHTML = "<div> front </div>";
    else if (color[2]==255) elt.innerHTML = "<div> back </div>";
        else elt.innerHTML = "<div> right </div>";
else if (color[1]==255)
    if (color[2]==255) elt.innerHTML = "<div> left </div>";
    else elt.innerHTML = "<div> top </div>";
else if (color[2]==255) elt.innerHTML = "<div> bottom </div>";
    else elt.innerHTML = "<div> background </div>";

gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```



Find which face was read and output it

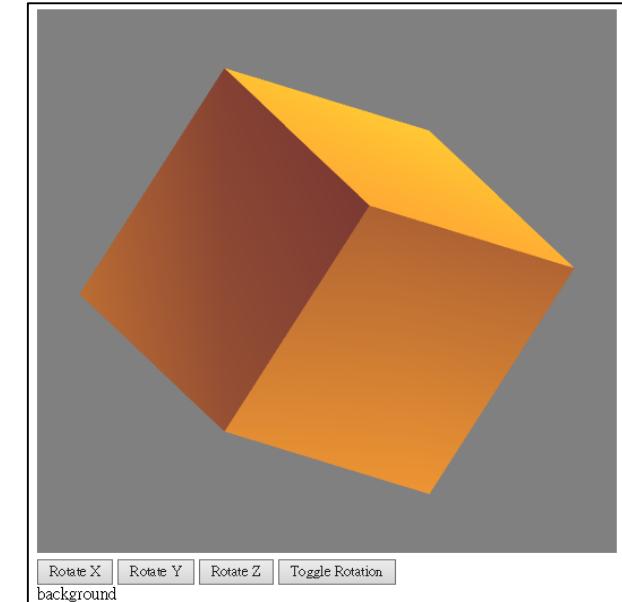
pickCube4.js (15/16)

```
gl.uniform1i(gl.getUniformLocation(program, "i"), 0);
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.uniform3fv(thetaLoc, theta);
    gl.drawArrays(gl.TRIANGLES, 0, 36);
}); // end of canvas.addEventListener("mousedown", function() {}

render();

} // end of window.onload
```

0: normal rendering



A cube has 6 faces and each face has 2 triangles.
Total vertices = 6 faces x 2 triangles/face x 3 vertices/triangle
= 36 vertices

pickCube4.js (16/16)

```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    if(flag) theta[axis] += 2.0;
    modelView = mat4();
    modelView = mult(modelView, rotate(theta[xAxis], [1, 0, 0] ));
    modelView = mult(modelView, rotate(theta[yAxis], [0, 1, 0] ));
    modelView = mult(modelView, rotate(theta[zAxis], [0, 0, 1] ));

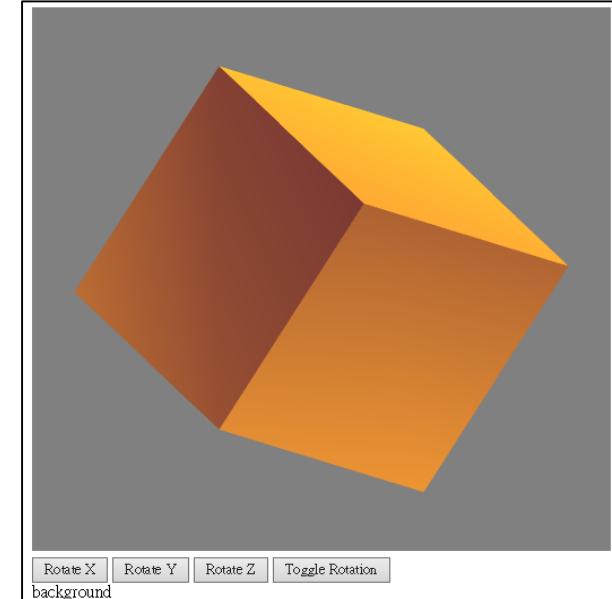
    gl.uniformMatrix4fv( gl.getUniformLocation(program, "modelViewMatrix"), false, flatten(modelView) );

    gl.uniform1i(gl.getUniformLocation(program, "i"), 0);
    gl.drawArrays( gl.TRIANGLES, 0, 36 );
}

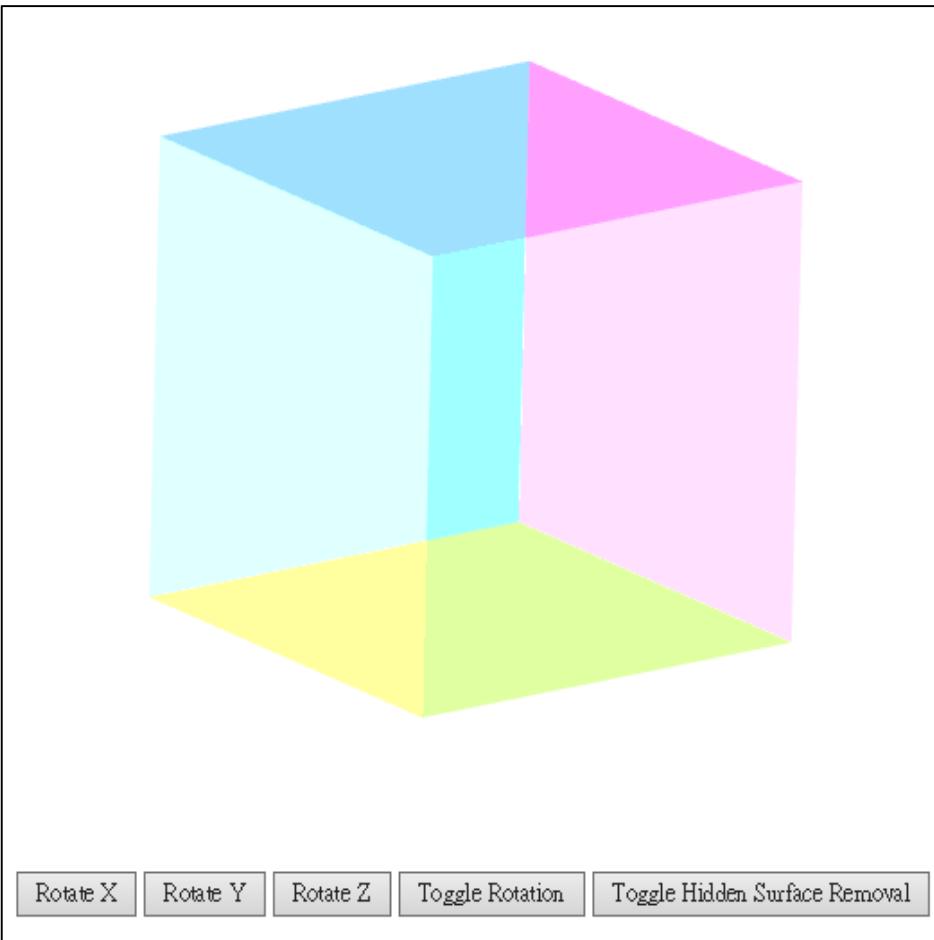
requestAnimFrame(render);
}
```

0: normal rendering

A cube has 6 faces and each face has 2 triangles.
Total vertices = 6 faces x 2 triangles/face x 3 vertices/triangle
= 36 vertices



Sample Programs: cubet.html, cubet.js



Rotating translucent cube.
Hidden-surface removal
can be toggled on and off.

cubet.html (1/4)

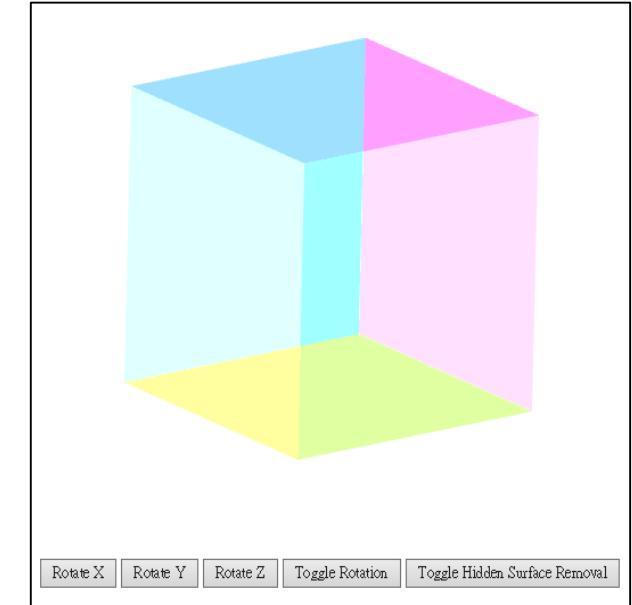
```
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;

uniform vec3 theta;

void main()
{
    // Compute the sines and cosines of theta for each of
    // the three axes in one computation.
    vec3 angles = radians( theta );
    vec3 c = cos( angles );
    vec3 s = sin ( angles );
}
```



cubet.html (2/4)

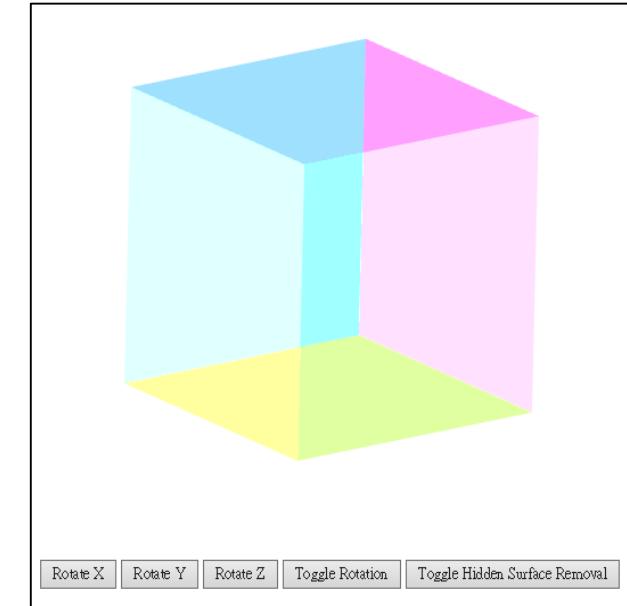
// Remember: these matrices are **column-major**

```
mat4 rx = mat4( 1.0, 0.0, 0.0, 0.0,
                  0.0, c.x, s.x, 0.0,
                  0.0, -s.x, c.x, 0.0,
                  0.0, 0.0, 0.0, 1.0 );
```

```
mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                  0.0, 1.0, 0.0, 0.0,
                  s.y, 0.0, c.y, 0.0,
                  0.0, 0.0, 0.0, 1.0 );
```

```
mat4 rz = mat4( c.z, -s.z, 0.0, 0.0,
                  s.z, c.z, 0.0, 0.0,
                  0.0, 0.0, 1.0, 0.0,
                  0.0, 0.0, 0.0, 1.0 );
```

```
fColor = vColor;
gl_Position = rz * ry * rx * vPosition;
}
</script>
```



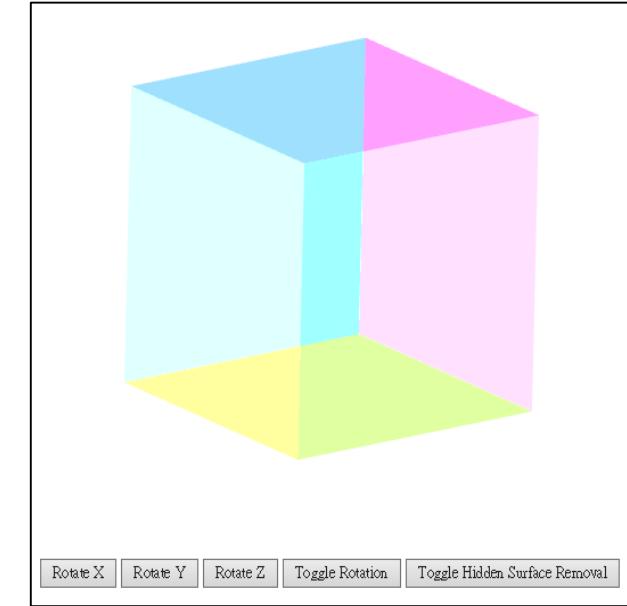
cubet.html (3/4)

```
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

varying vec4 fColor;

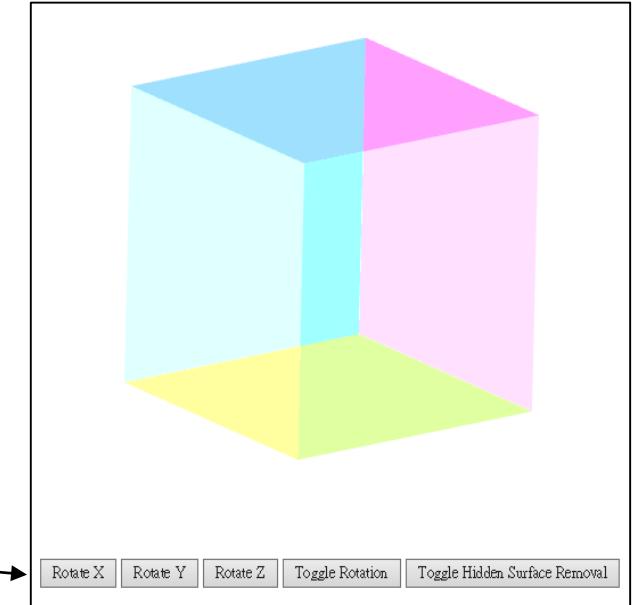
void main()
{
    gl_FragColor = fColor;
}
</script>
```

```
<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="cubet.js"></script>
```



cubet.html (4/4)

```
<body>  
<canvas id="gl-canvas" width="512" height="512">  
Oops ... your browser doesn't support the HTML5 canvas element  
</canvas>  
  
<br/>  
  
<button id = "ButtonX">Rotate X</button>  
<button id = "ButtonY">Rotate Y</button>  
<button id = "ButtonZ">Rotate Z</button>  
<button id = "ButtonT">Toggle Rotation</button>  
<button id = "ButtonH">Toggle Hidden Surface Removal</button>  
  
</body>  
</html>
```



cubet.js (1/10)

```
var canvas;
var gl;

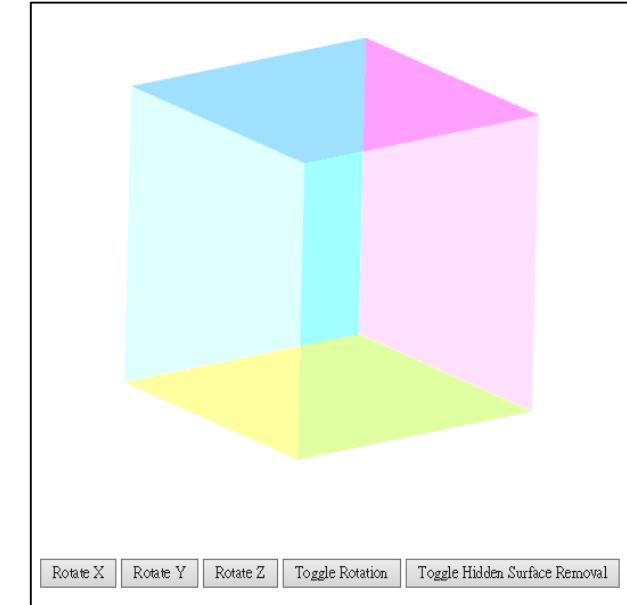
var numVertices = 36;

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

var xAxis = 0;
var yAxis = 1;
var zAxis = 2;
var axis = xAxis;

var flag = true;
var HSRflag = true;

var theta = [ 45.0, 45.0, 45.0 ];
var thetaLoc;
```

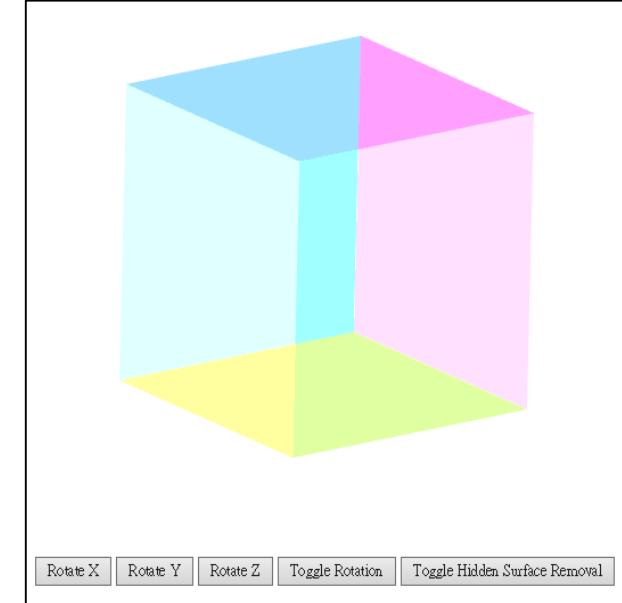


cubet.js (2/10)

```
window.onload = function ()  
{  
    canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    colorCube();  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );  
  
    gl.enable(gl.DEPTH_TEST);  
    gl.enable(gl.BLEND);  
    gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
```

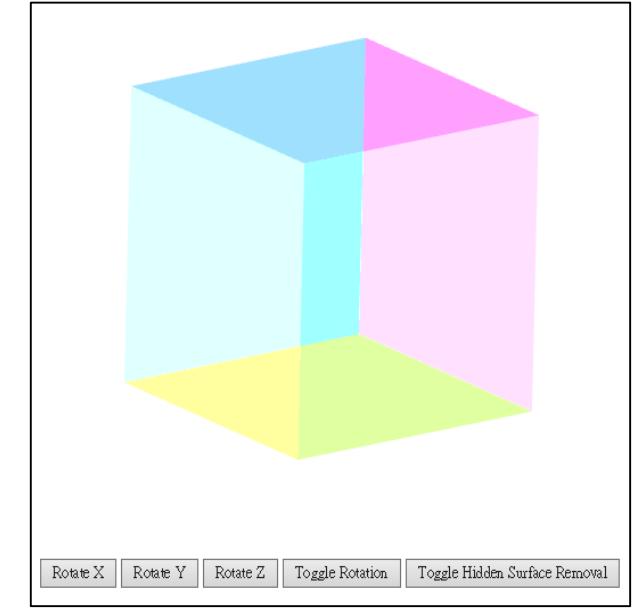
Source blending factor

Destination blending factor



cubet.js (3/10)

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



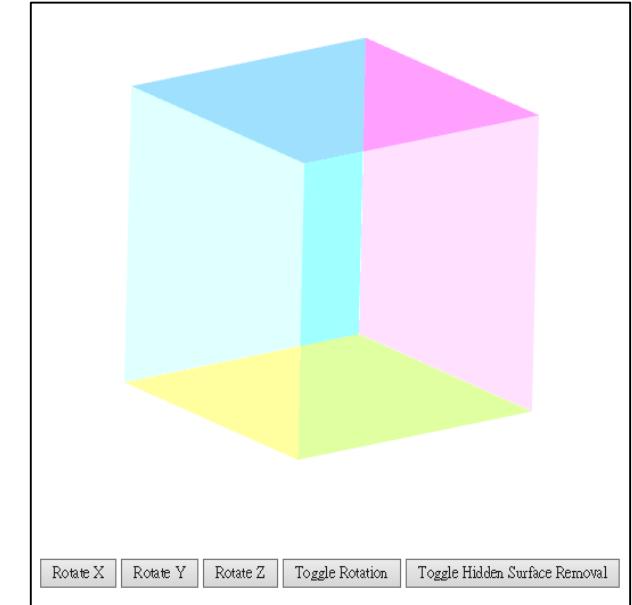
cubet.js (4/10)

```
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 );

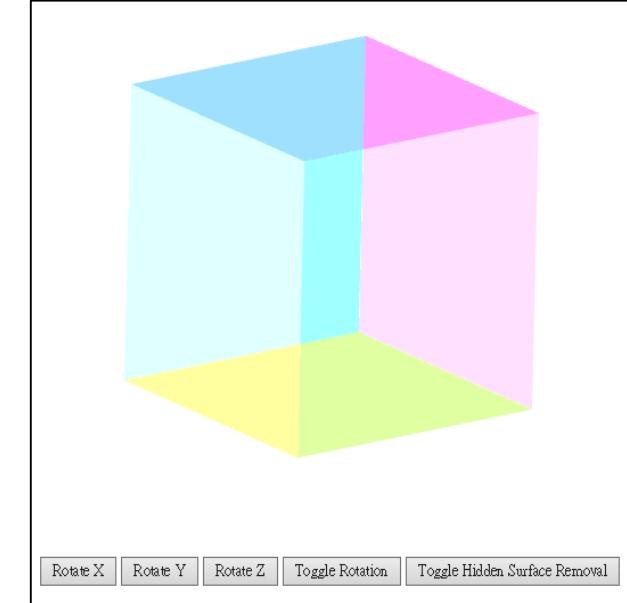
thetaLoc = gl.getUniformLocation(program, "theta");

document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};
document.getElementById("ButtonT").onclick = function() {flag = !flag;};
```



cubet.js (5/10)

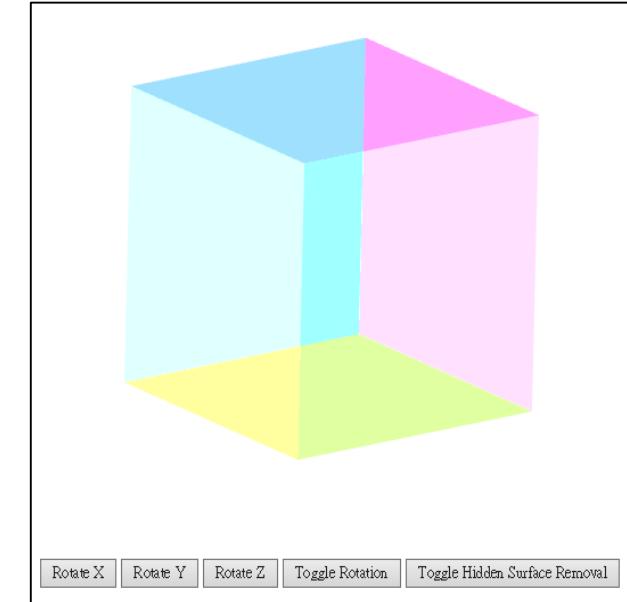
```
document.getElementById("ButtonH").onclick = function() {  
  
    if (HSRflag) gl.enable(gl.DEPTH_TEST);  
    else      gl.disable(gl.DEPTH_TEST);  
    HSRflag = !HSRflag;  
  
}; // end of document.getElementById("ButtonH").onclick =function () {}  
  
render();  
  
} // end of window.onload
```



Rotate X Rotate Y Rotate Z Toggle Rotation Toggle Hidden Surface Removal

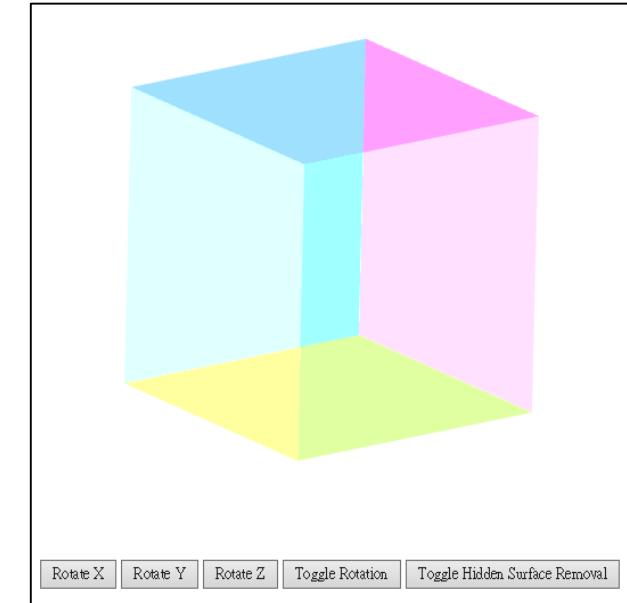
cubet.js (6/10)

```
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 );
}
```



cubet.js (7/10)

```
function quad(a, b, c, d)
{
    var vertices = [
        vec3( -0.5, -0.5, 0.5 ),
        vec3( -0.5, 0.5, 0.5 ),
        vec3( 0.5, 0.5, 0.5 ),
        vec3( 0.5, -0.5, 0.5 ),
        vec3( -0.5, -0.5, -0.5 ),
        vec3( -0.5, 0.5, -0.5 ),
        vec3( 0.5, 0.5, -0.5 ),
        vec3( 0.5, -0.5, -0.5 )
    ];
```

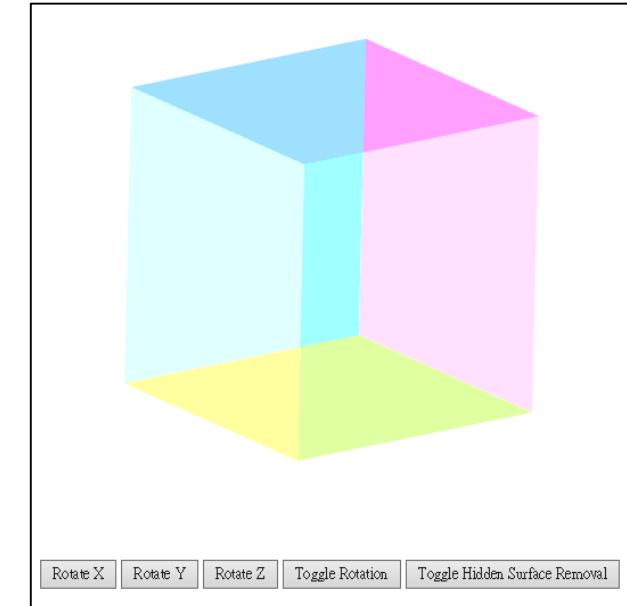


cubet.js (8/10)

```
var vertexColors = [  
    [ 0.0, 0.0, 0.0, 0.5 ], // black  
    [ 1.0, 0.0, 0.0, 0.5 ], // red  
    [ 1.0, 1.0, 0.0, 0.5 ], // yellow  
    [ 0.0, 1.0, 0.0, 0.5 ], // green  
    [ 0.0, 0.0, 1.0, 0.5 ], // blue  
    [ 1.0, 0.0, 1.0, 0.5 ], // magenta  
    [ 0.0, 1.0, 1.0, 0.5 ], // cyan  
    [ 1.0, 1.0, 1.0, 0.5 ] // white  
];
```



α value



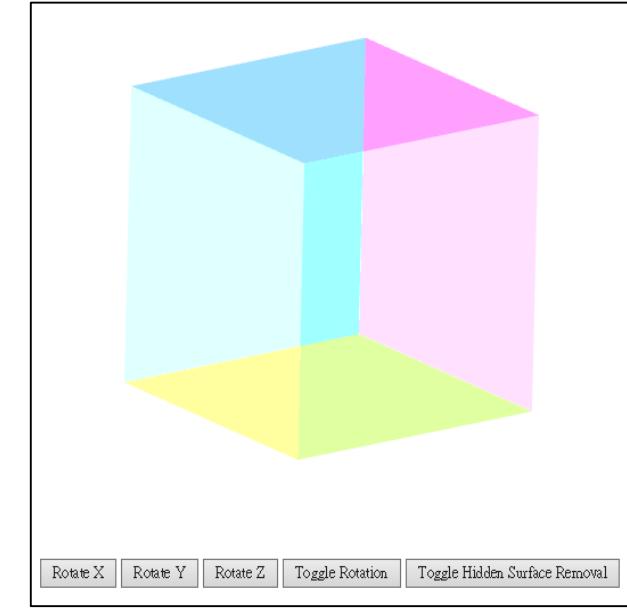
cubet.js (9/10)

```
// We need to partition the quad into two triangles in order for
// WebGL to be able to render it. In this case, we create two
// triangles from the quad.

var indices = [ a, b, c, a, c, d ];

for ( var i = 0; i < indices.length; ++i ) {
    points.push( vertices[indices[i]] );
    colors.push( vertexColors[a] );
}

} // end of quad(a, b, c, d)
```



cubet.js (10/10)

```
function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    if(flag) theta[axis] += 2.0;
    gl.uniform3fv(thetaLoc, theta);

    gl.drawArrays( gl.TRIANGLES, 0, numVertices );

    requestAnimFrame( render );
}

}
```

