5. Viewing

Outline

- Classical Viewing
- Computer Viewing
  - Positioning the Camera
  - Projection
- Orthogonal Projection Matrices
- Perspective Projection Matrices
- Meshes
- Shadows
- Sample Programs

Classical Viewing

Objectives

- Introduce the classical views
- Compare and contrast image formation by computer with how images have been formed by architects, artists, and engineers
- Learn the benefits and drawbacks of each type of view

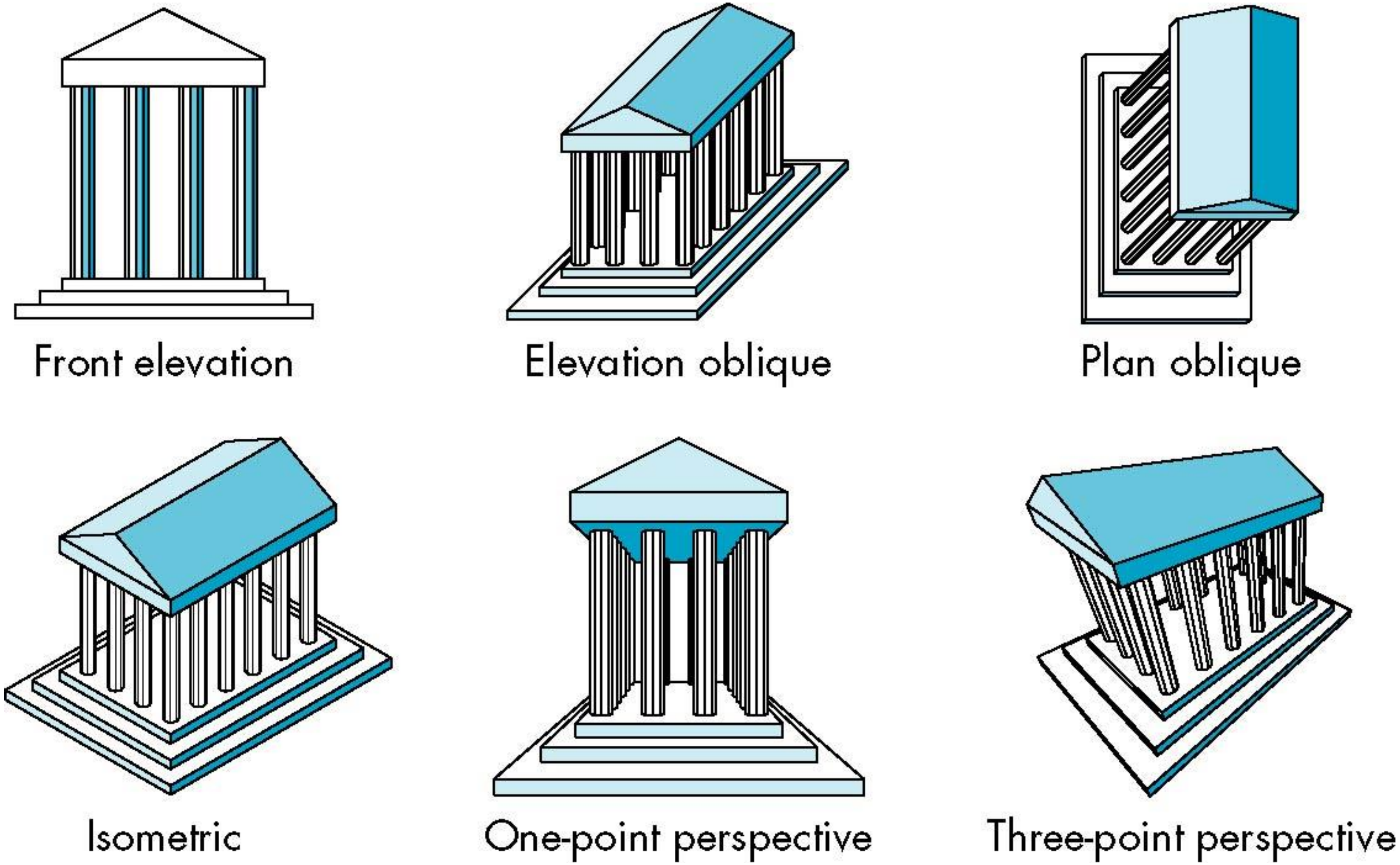
Classical Viewing

- Viewing requires three basic elements
  - One or more objects
  - A viewer with a projection surface
  - Projectors that go from the object(s) to the projection surface
- Classical views are based on the relationship among these elements
  - The viewer picks up the object and orients it how she would like to see it
- Each object is assumed to constructed from flat *principal faces*
  - Buildings, polyhedra, manufactured objects

Planar Geometric Projections

- Standard projections project onto a plane
- Projectors are lines that either
  - converge at a center of projection
  - are parallel
- Such projections preserve lines
  - but not necessarily angles
- Nonplanar projections are needed for applications such as map construction

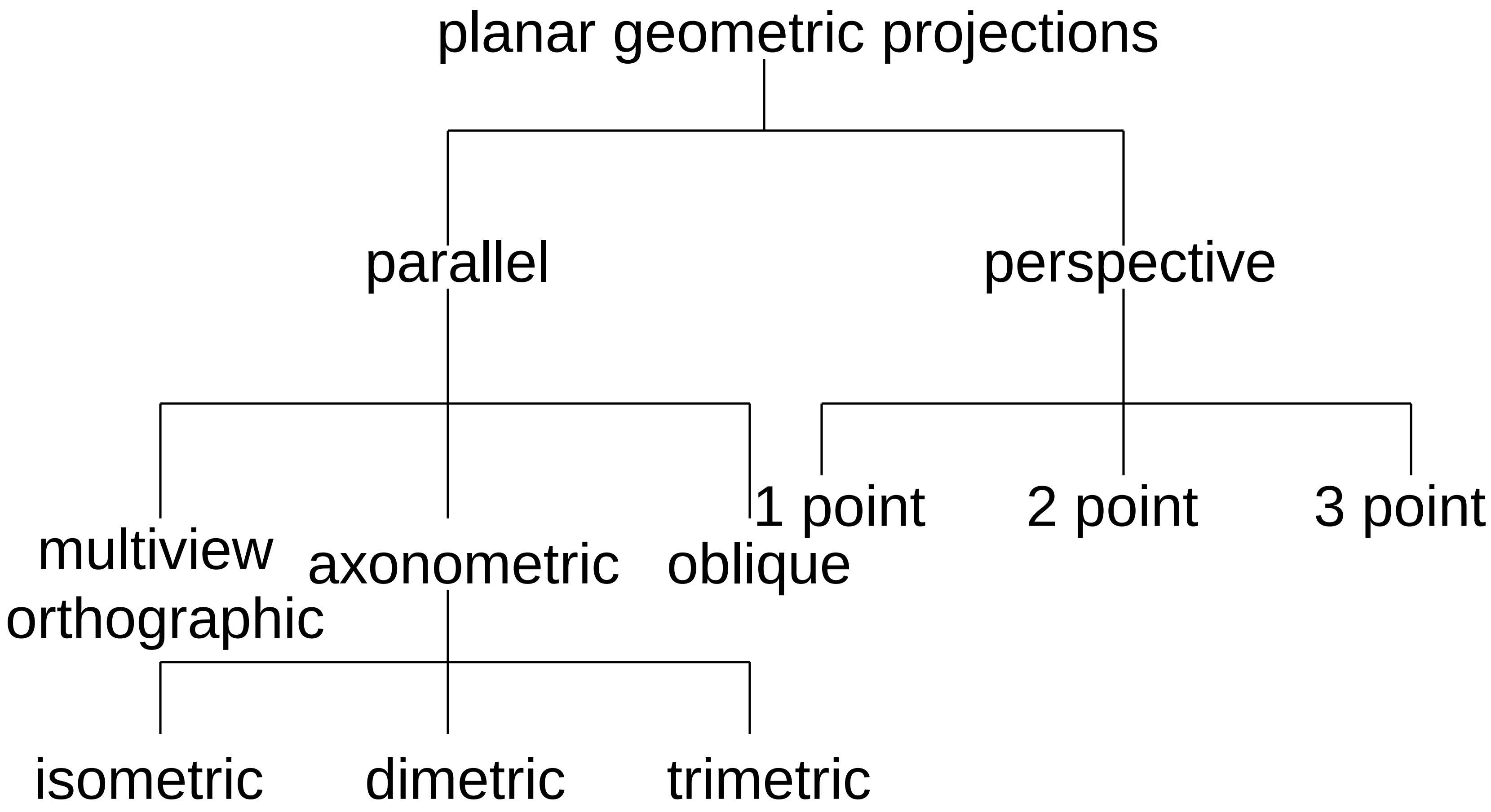
Classical Projections



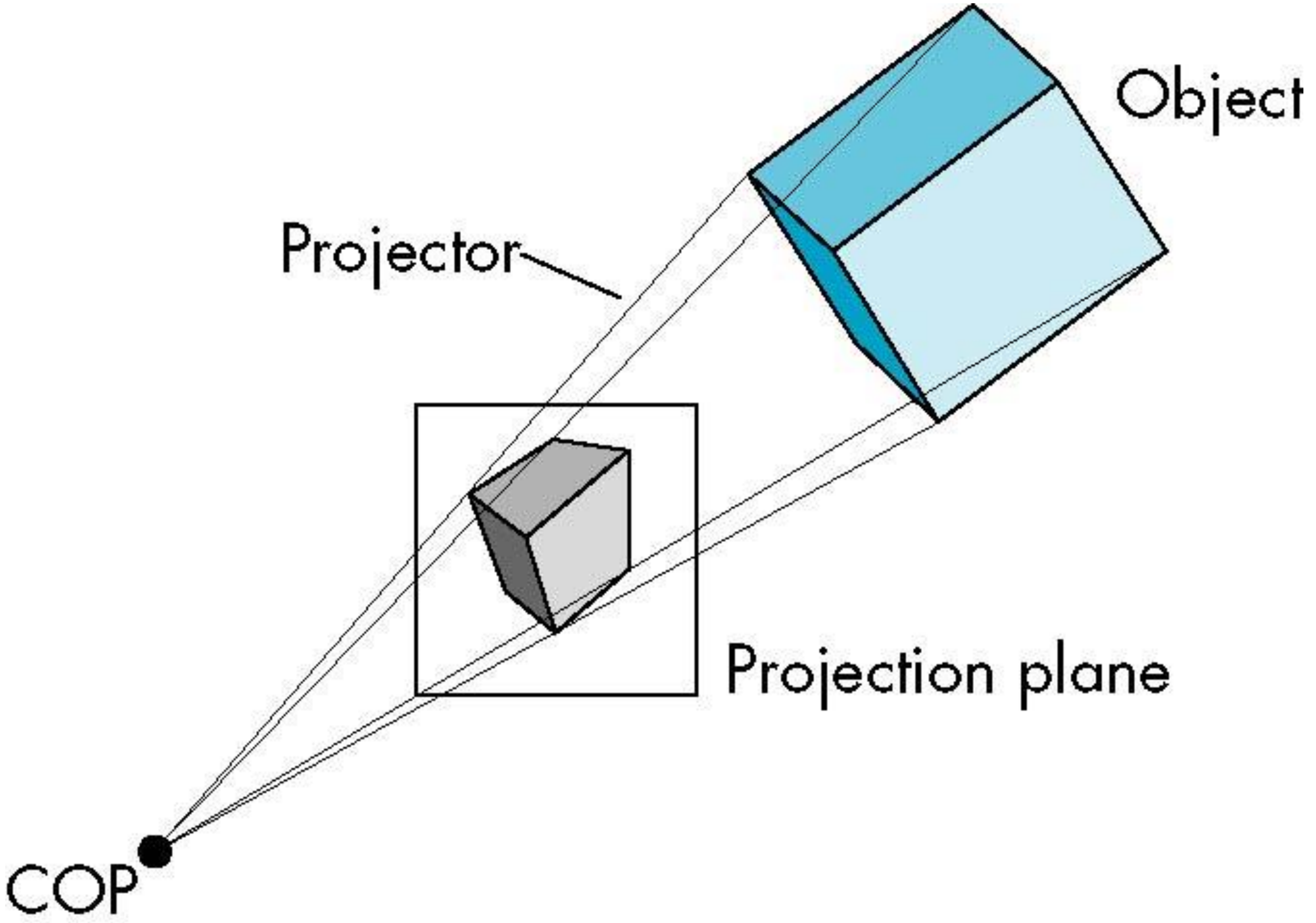
Perspective vs Parallel

- Computer graphics treats all projections the same and implements them with a single pipeline
- Classical viewing developed different techniques for drawing each type of projection
- Fundamental distinction is between parallel and perspective viewing even though mathematically parallel viewing is the limit of perspective viewing

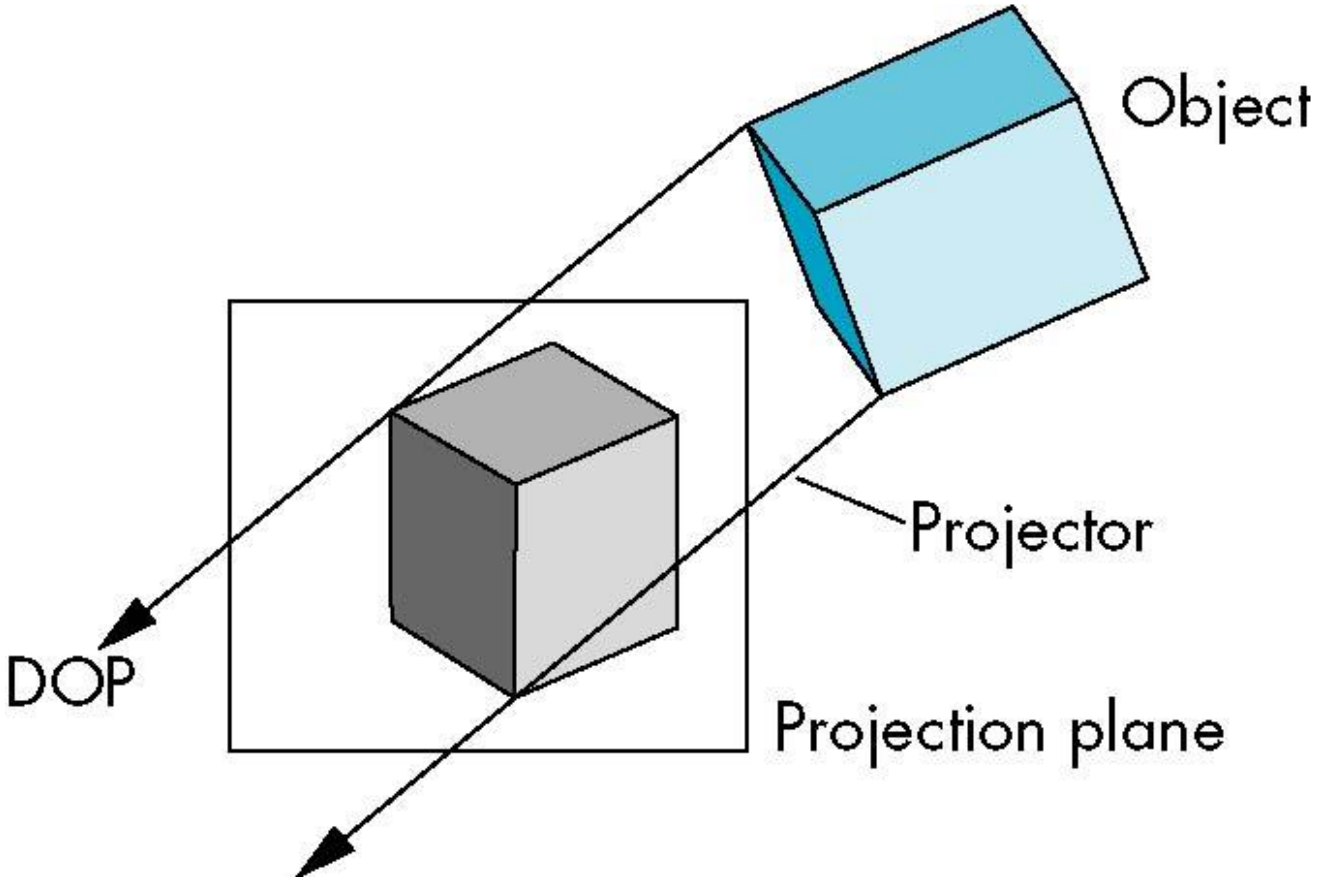
Taxonomy of Planar Geometric Projections



Perspective Projection

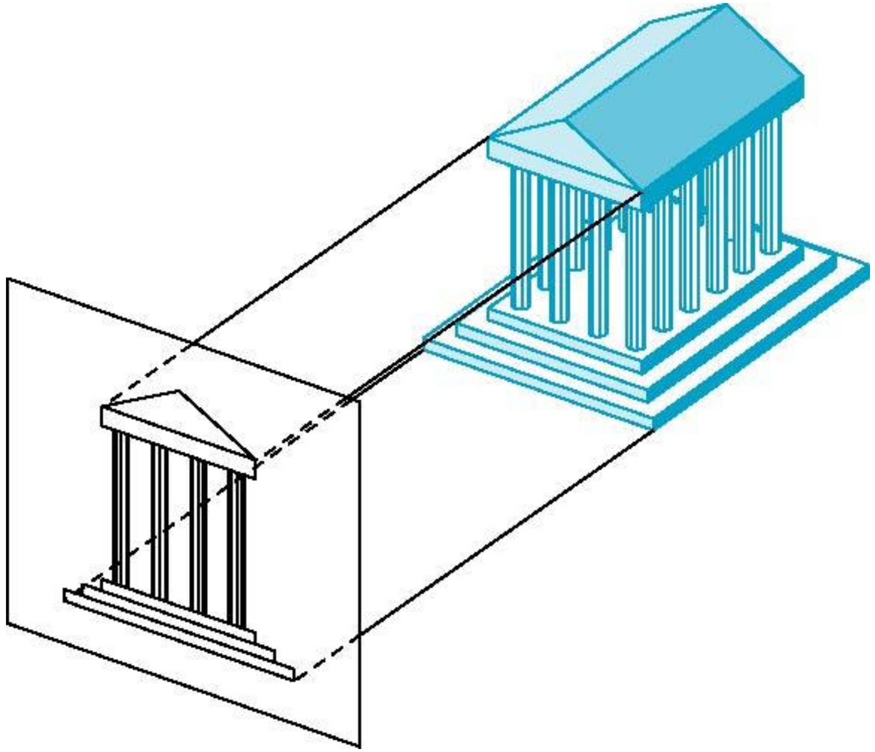


Parallel Projection



Orthographic Projection

Projectors are orthogonal to projection surface



Multiview Orthographic Projection

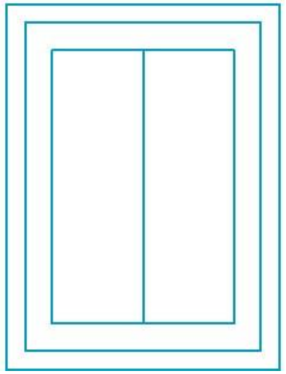
- Projection plane parallel to principal face
- Usually form front, top, side views

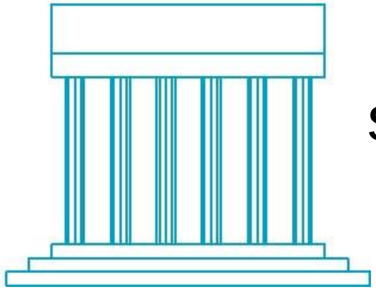
isometric (not multiview
orthographic view)

front

in CAD and architecture,
we often display three
multiviews plus isometric

top





side

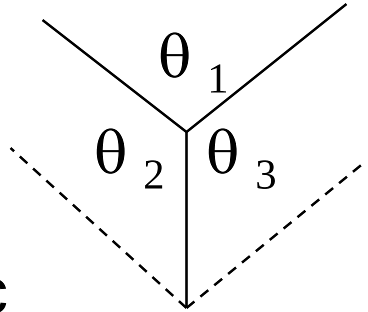
Advantages and Disadvantages

- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals
- Cannot see what object really looks like because many surfaces hidden from view
  - Often we add the isometric

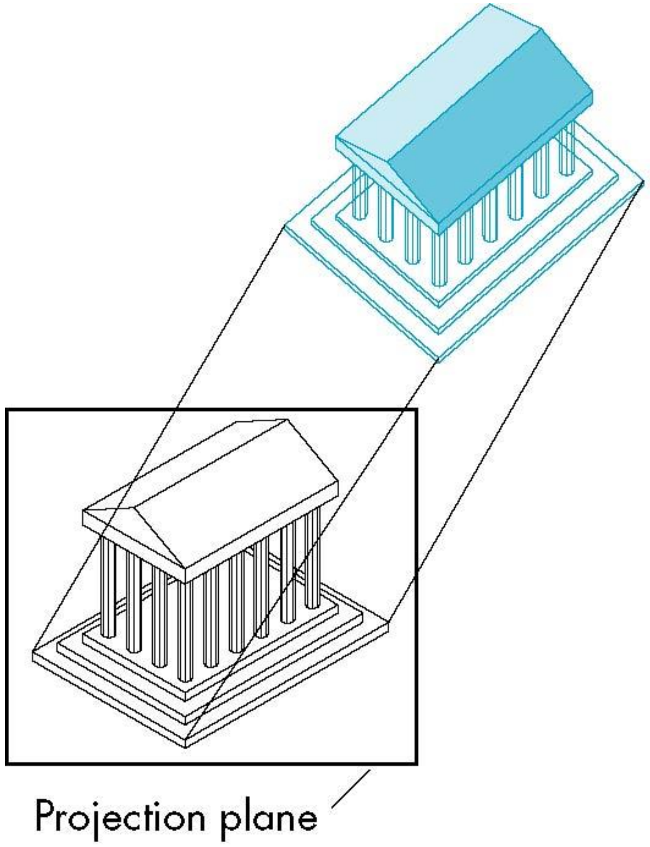
Axonometric Projections

Allow projection plane to move relative to object

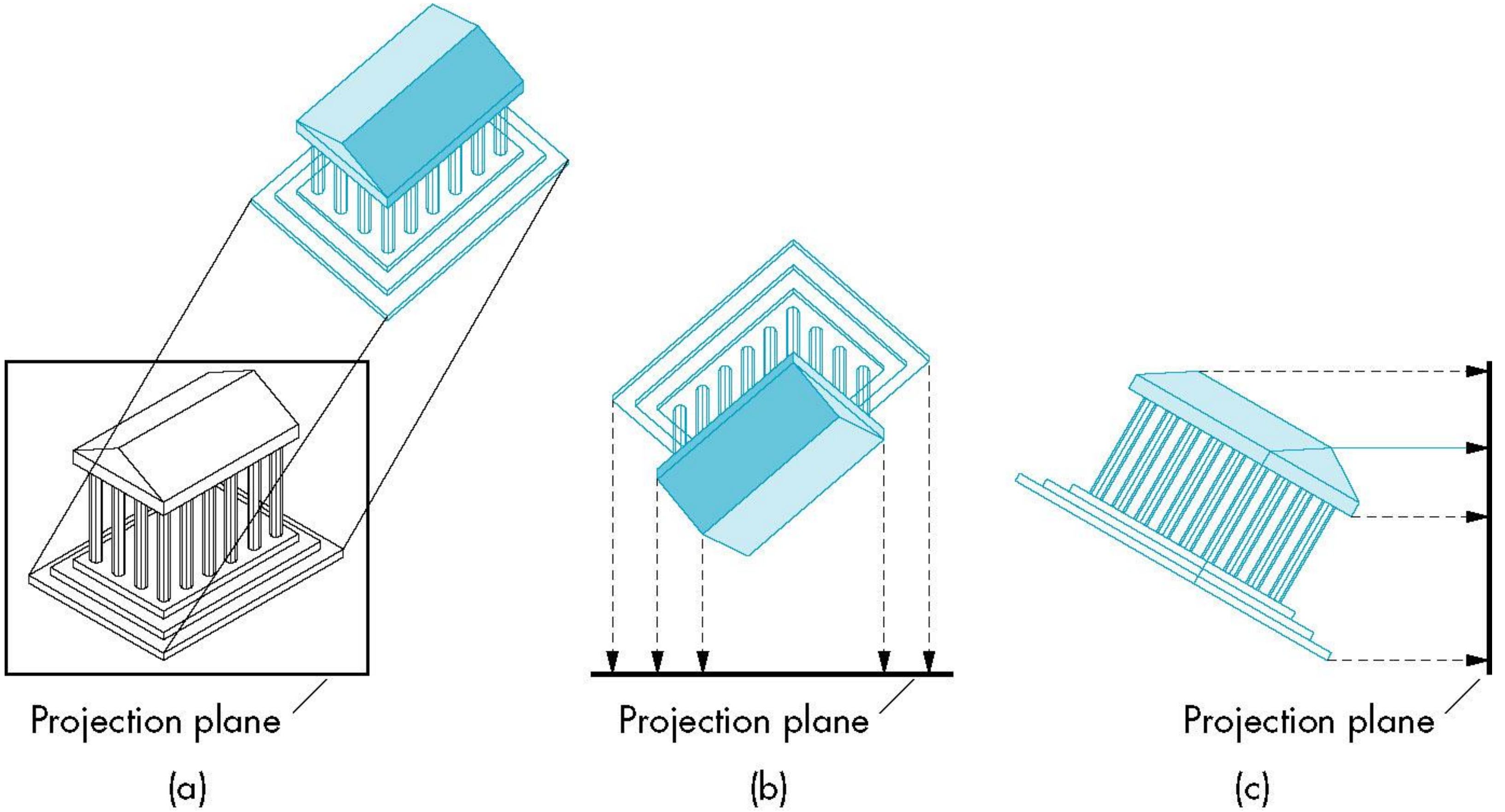
classify by how many angles of a corner of a projected cube are the same



none: trimetric
two: dimetric
three: isometric



Axonometric Projections

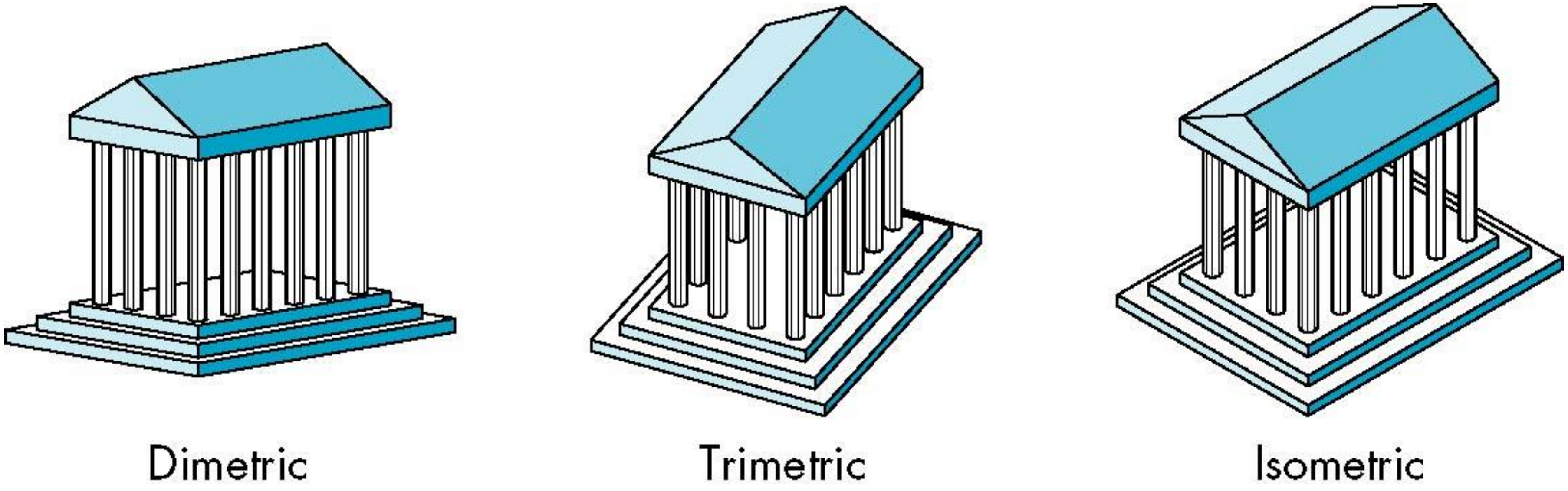


Construction of trimetric-view projection

Top view

Side view

Types of Axonometric Projections

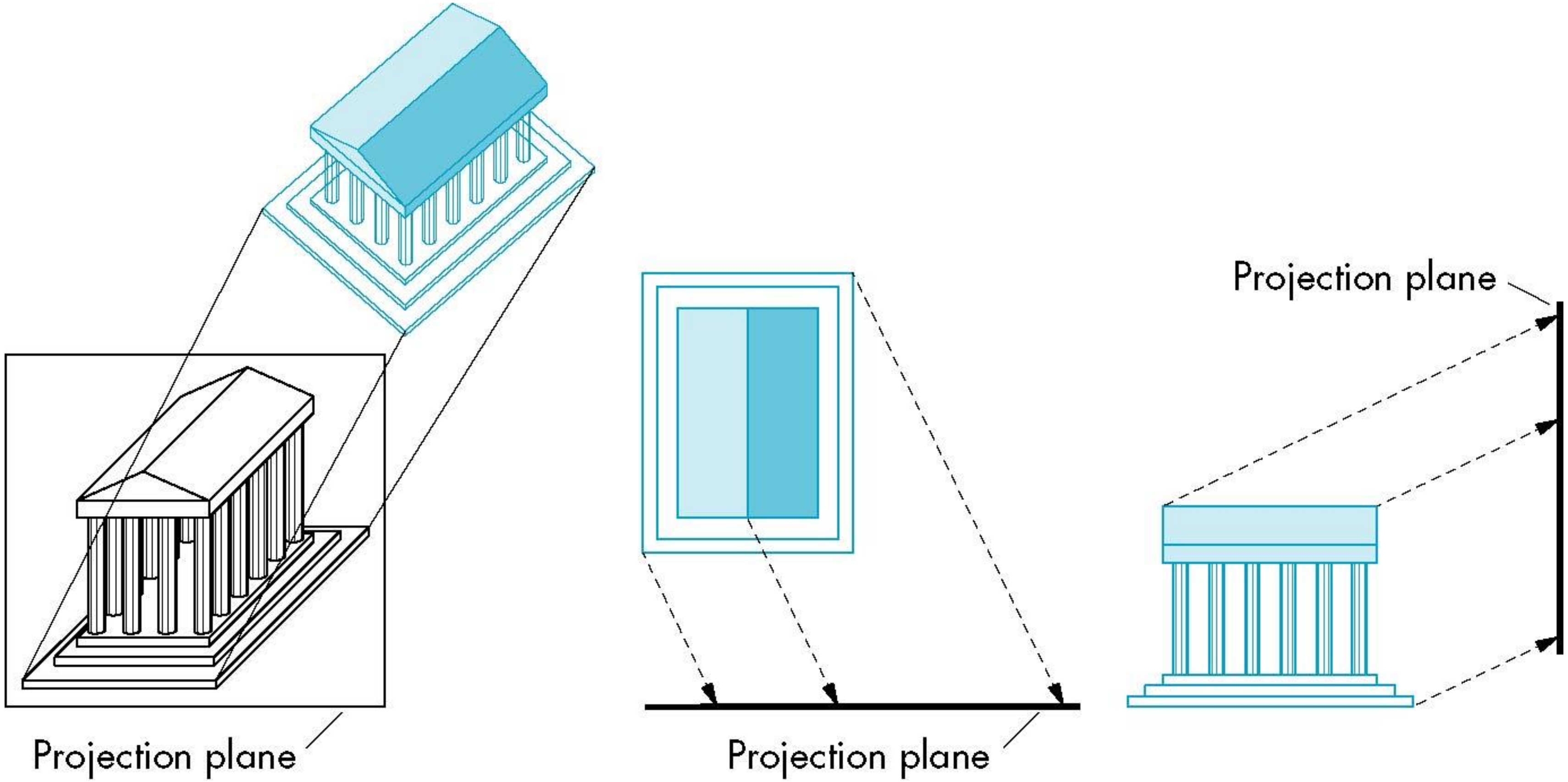


Advantages and Disadvantages

- Lines are scaled (*foreshortened*) but can find scaling factors
- Lines preserved but angles are not
  - Projection of a circle in a plane not parallel to the projection plane is an ellipse
- Can see three principal faces of a box-like object
- Some optical illusions possible
  - Parallel lines appear to diverge
- Does not look real because far objects are scaled the same as near objects
- Used in CAD applications

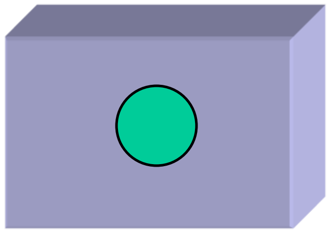
Oblique Projection

Arbitrary relationship between projectors and projection plane



Advantages and Disadvantages

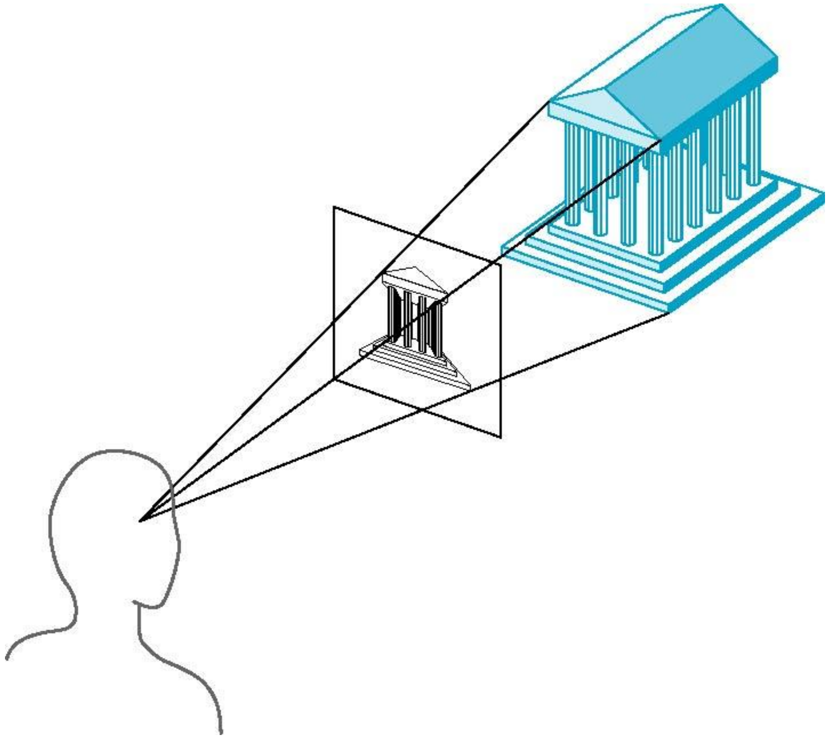
- Can pick the angles to emphasize a particular face
  - Architecture: plan oblique, elevation oblique
- Angles in faces parallel to projection plane are preserved while we can still see “around” side



- In physical world, cannot create with simple camera; possible with bellows camera or special lens (architectural)

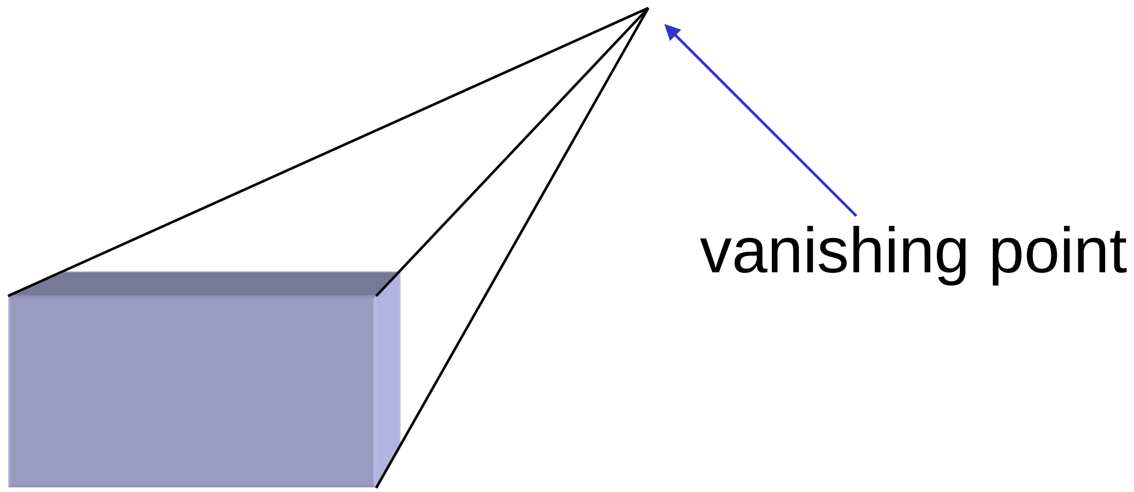
Perspective Projection

Projectors coverge at center of projection



Vanishing Points

- Parallel lines (not parallel to the projection plan) on the object converge at a single point in the projection (the *vanishing point*)
- Drawing simple perspectives by hand uses these vanishing point(s)



Three-Point Perspective

- No principal face parallel to projection plane
- Three vanishing points for cube

Two-Point Perspective

- On principal direction parallel to projection plane
- Two vanishing points for cube

One-Point Perspective

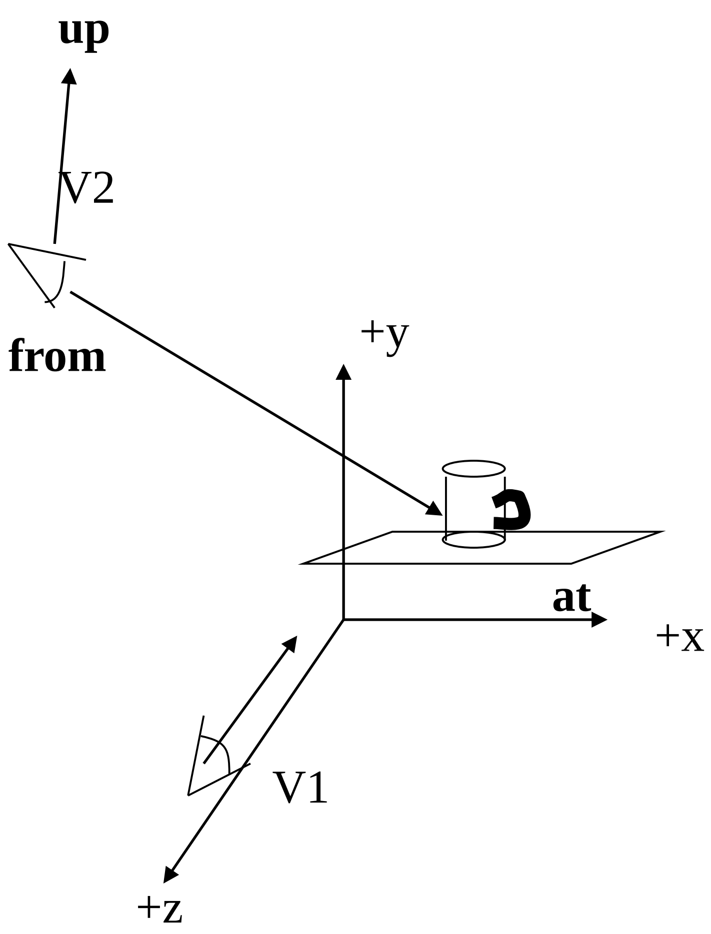
- One principal face parallel to projection plane
- One vanishing point for cube

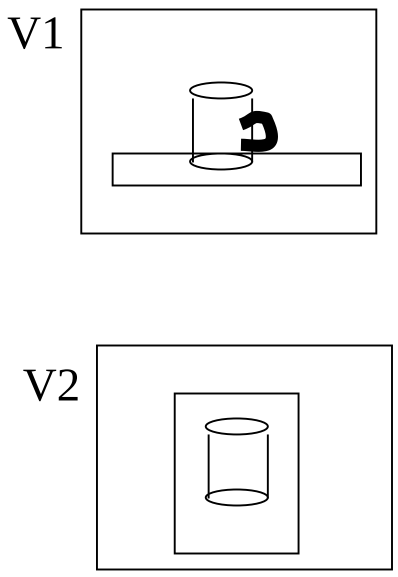
Advantages and Disadvantages

- Objects further from viewer are projected smaller than the same sized objects closer to the viewer (*diminution*)
  - Looks realistic
- Equal distances along a line are not projected into equal distances (*nonuniform foreshortening*)
- Angles preserved only in planes parallel to the projection plane
- More difficult to construct by hand than parallel projections (but not more difficult by computer)

Computer Viewing
Positioning the Camera

Viewing Transformation





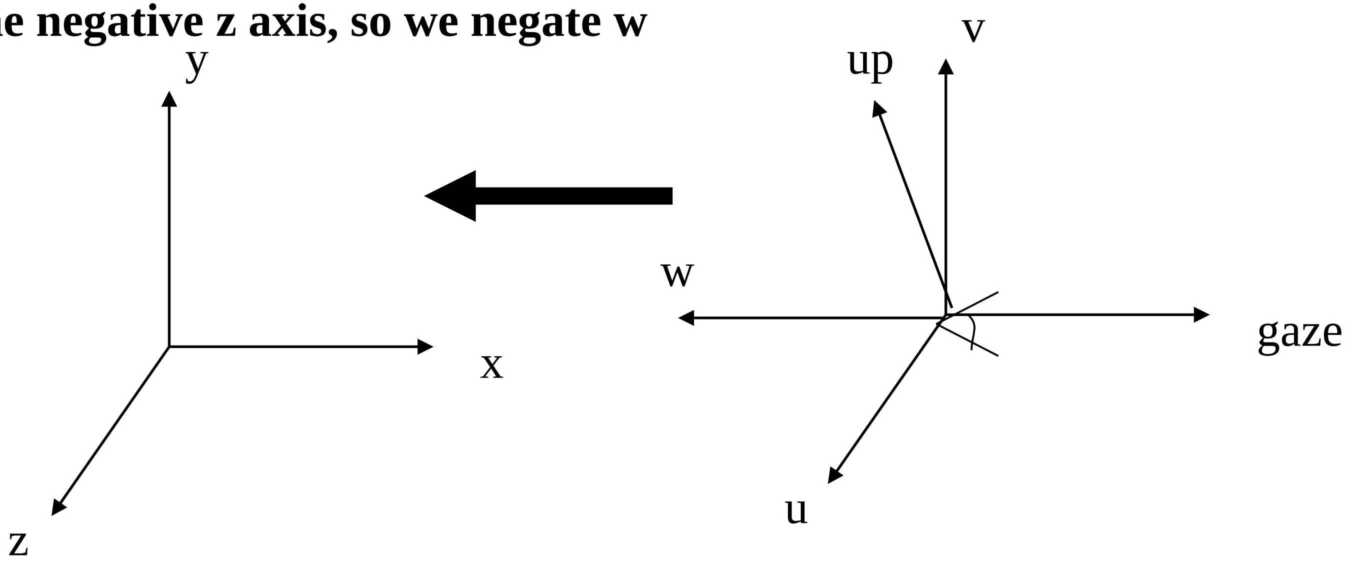
How do we get from V2 to V1?

Viewing Transformation

Fill the rows of the rotation matrix: R

$$w = -\frac{at - from}{\left\| at - from \right\|} \qquad u = \frac{up \times w}{\left\| up \times w \right\|} \qquad v = w \times u$$

Note: This will orient the eye looking down the z axis!!! We want it looking down the negative z axis, so we negate w



Graphics Pipeline

Model Space

Model Transformations ↓

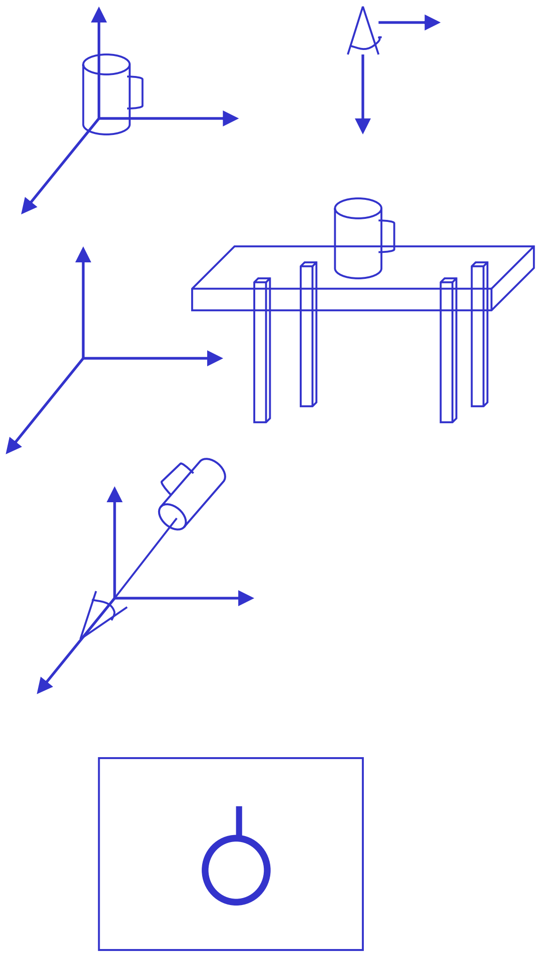
World Space

Viewing Transformation ↓

Eye/Camera Space

Projection & Window Transformation ↓

Screen Space



Graphics Pipeline

(ortho projection only)

Model Space

M_{model} **Model Transformations**

World Space

$M_{viewing}$ **Viewing Transformation**

Eye/Camera Space

$M_{orthographic}$ **Projection & Window Transformation**

(ortho projection includes the windowing transform)

Screen Space

$$P' = M_{ortho} M_{viewing} M_{model} P$$

Viewing

- Orthographic views
- Perspective views

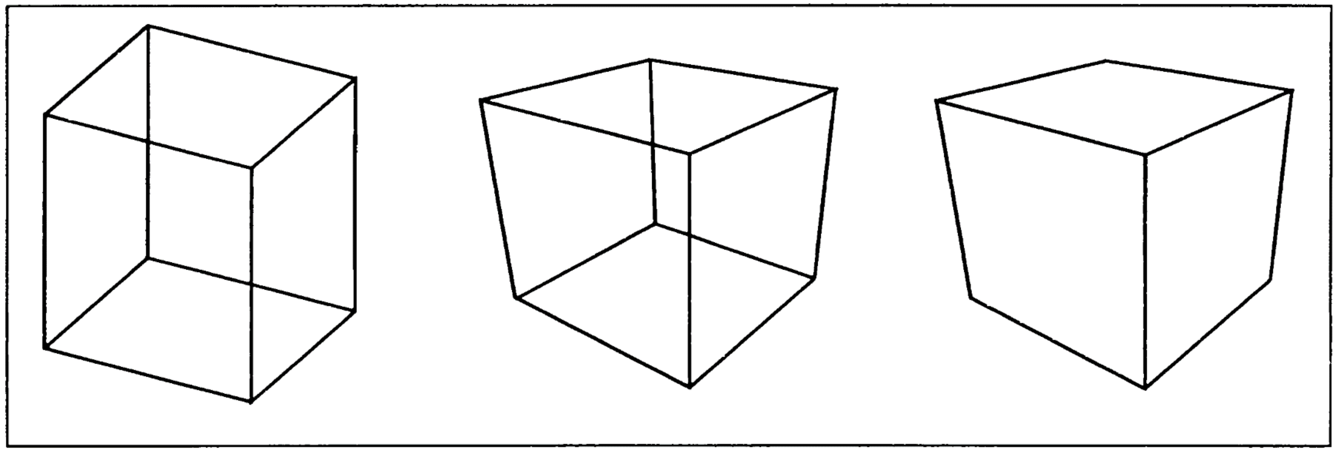
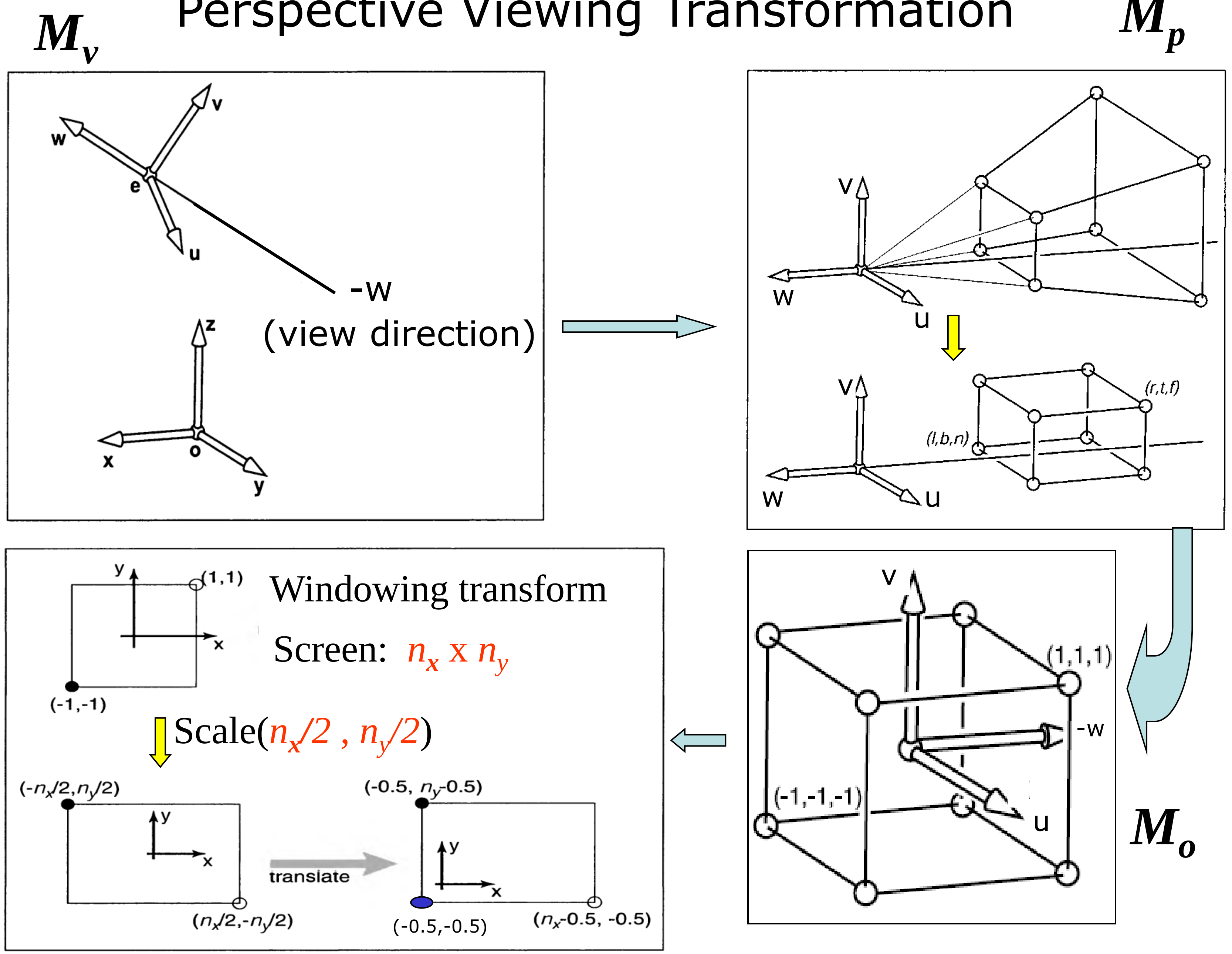


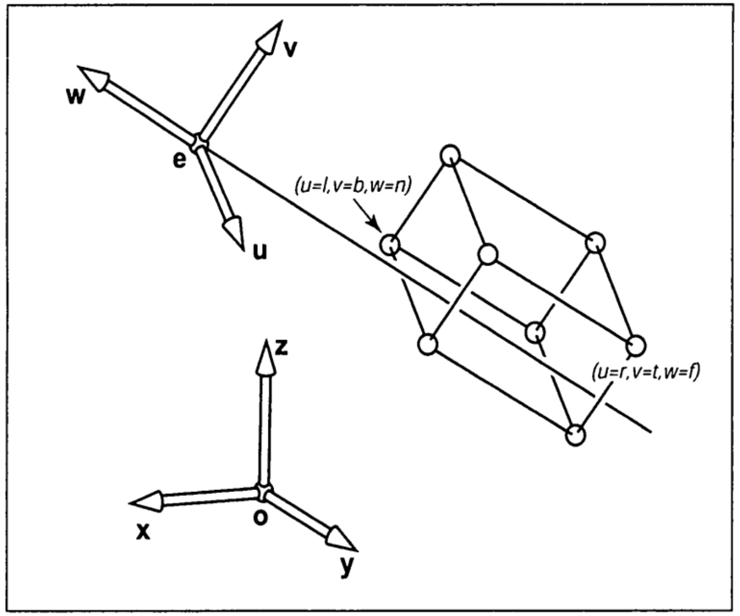
Figure 7.1. Left: orthographic projection. Middle: perspective projection. Right: perspective projection with hidden lines removed.

Perspective Viewing Transformation

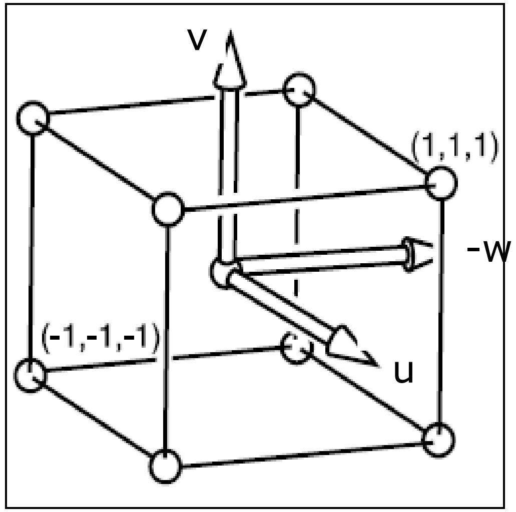


Orthographic Viewing Transformation

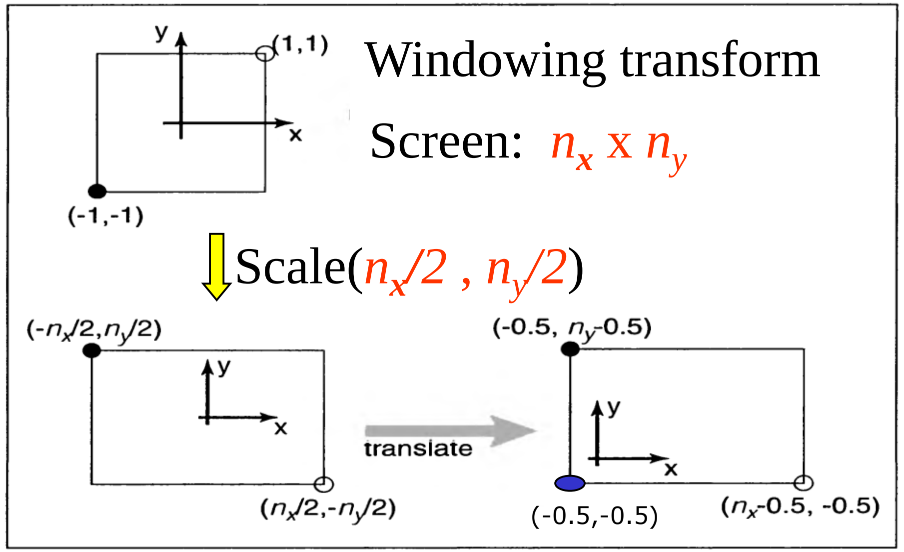
M_v



$$M = M_o M_v$$



M_o



Objectives

- Introduce the mathematics of projection
- Introduce WebGL viewing functions in MV.js
- Look at alternate viewing APIs

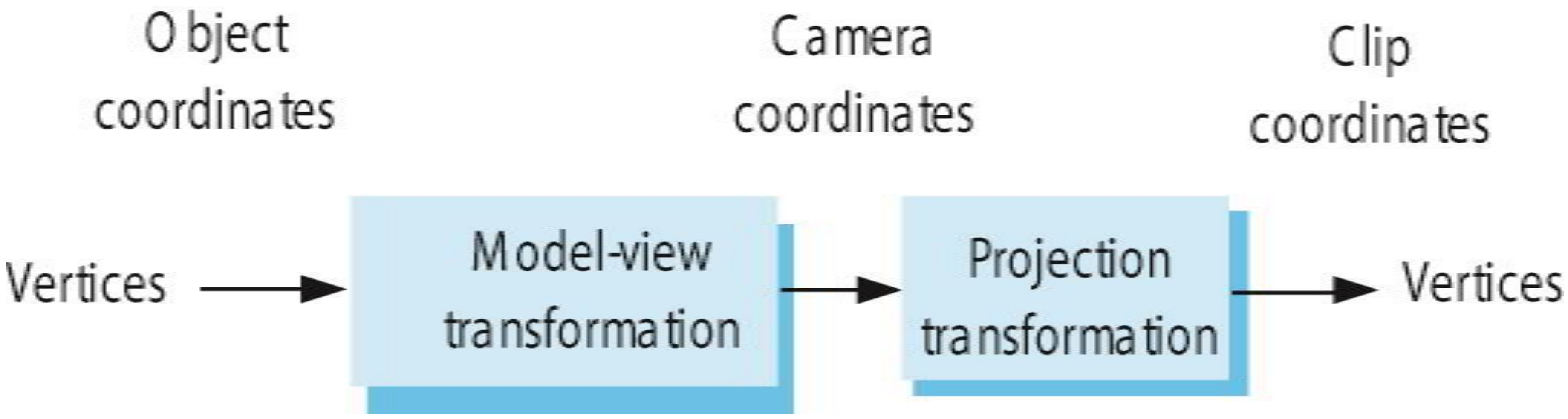
From the Beginning

- In the beginning:
  - fixed function pipeline
  - Model-View and Projection Transformation
  - Predefined frames: model, object, camera, clip, ndc, window
- After deprecation
  - pipeline with programmable shaders
  - no transformations
  - clip, ndc window frames
- MV.js reintroduces original capabilities

Computer Viewing

- There are three aspects of the viewing process, all of which are implemented in the pipeline,
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

Viewing Transformation

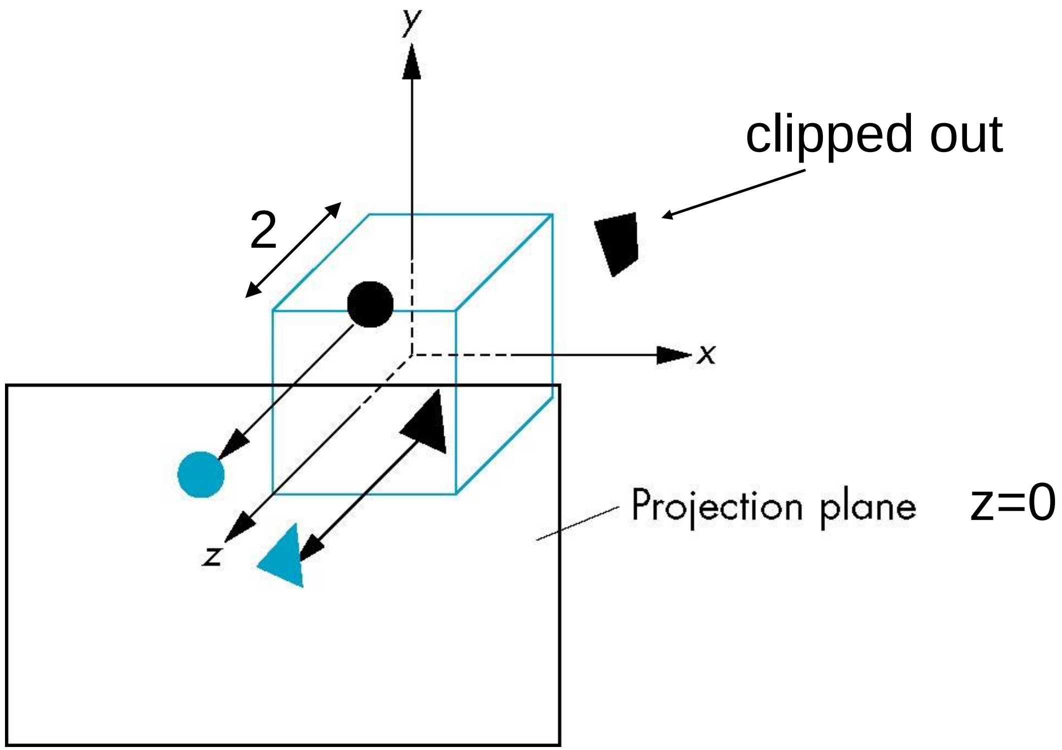


The WebGL Camera

- In WebGL, initially the object and camera frames are the same
  - Default model-view matrix is an identity
- The camera is located at origin and points in the negative z direction
- WebGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
  - Default projection matrix is an identity

Default Projection

Default projection is orthogonal



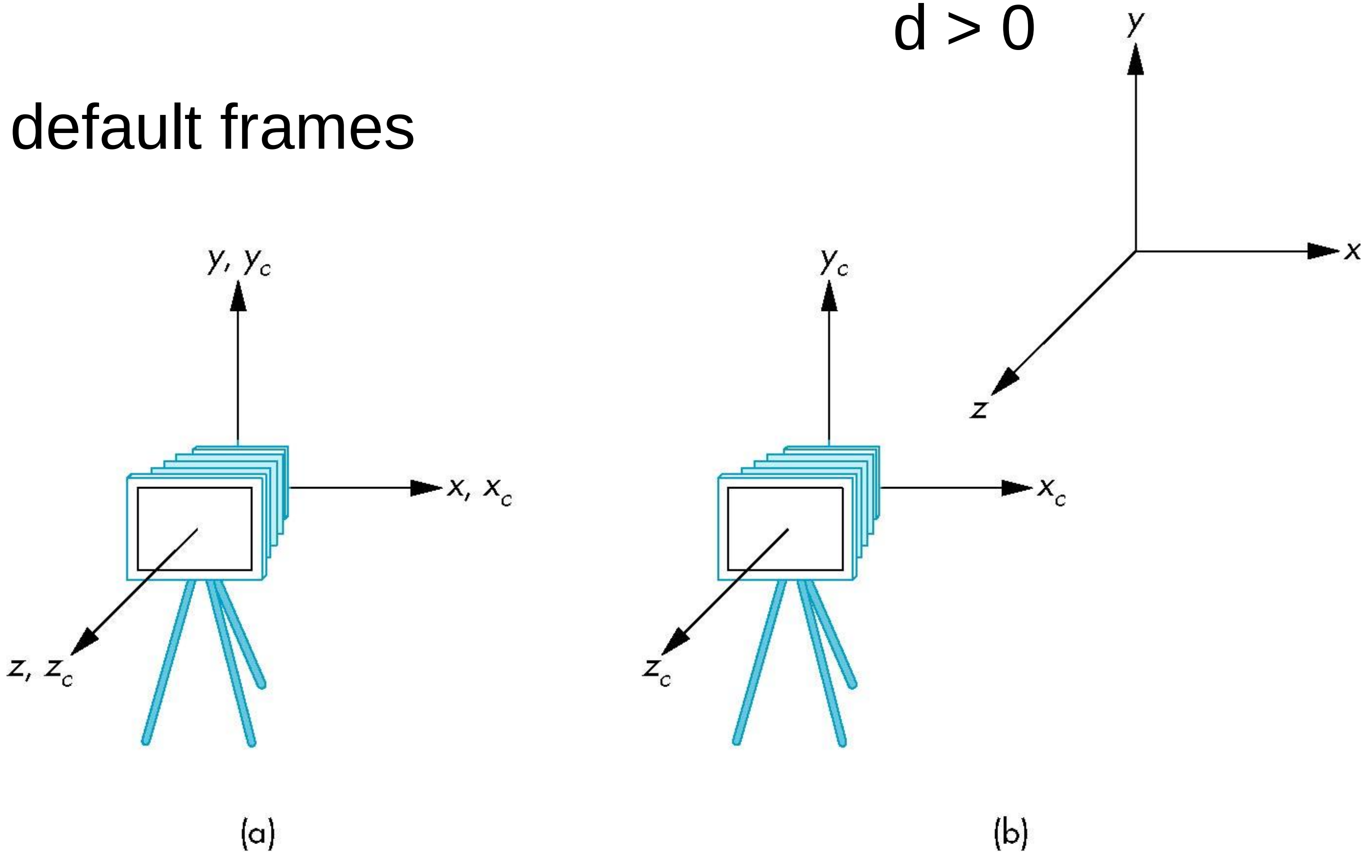
Moving the Camera Frame

- If we want to visualize objects with both positive and negative z values we can either
  - Move the camera in the positive z direction
    - Translate the camera frame
  - Move the objects in the negative z direction
    - Translate the world frame
- Both of these views are equivalent and are determined by the model-view matrix
  - Want a translation (`translate(0.0,0.0,-d);`)
  - `-d > 0`

Moving Camera back from Origin

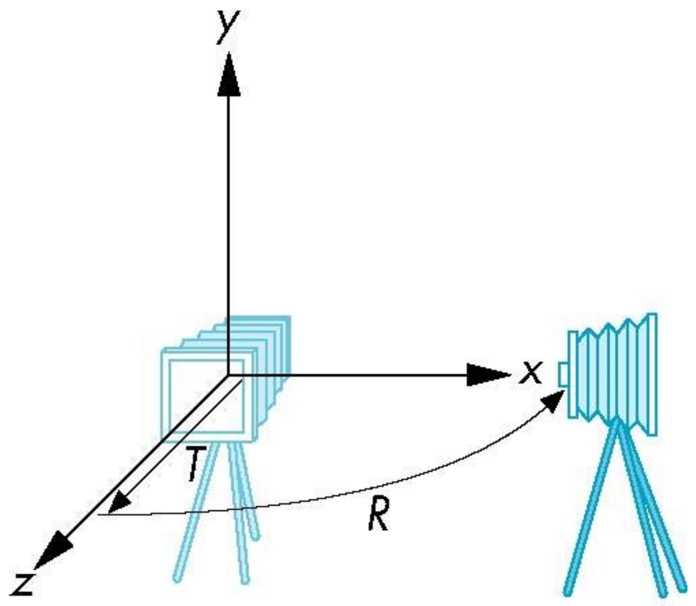
frames after translation by –d
d > 0

default frames



Moving the Camera

- We can move the camera to any desired position by a sequence of rotations and translations
- Example: side view
  - Rotate the camera
  - Move it away from origin
  - Model-view matrix C = TR



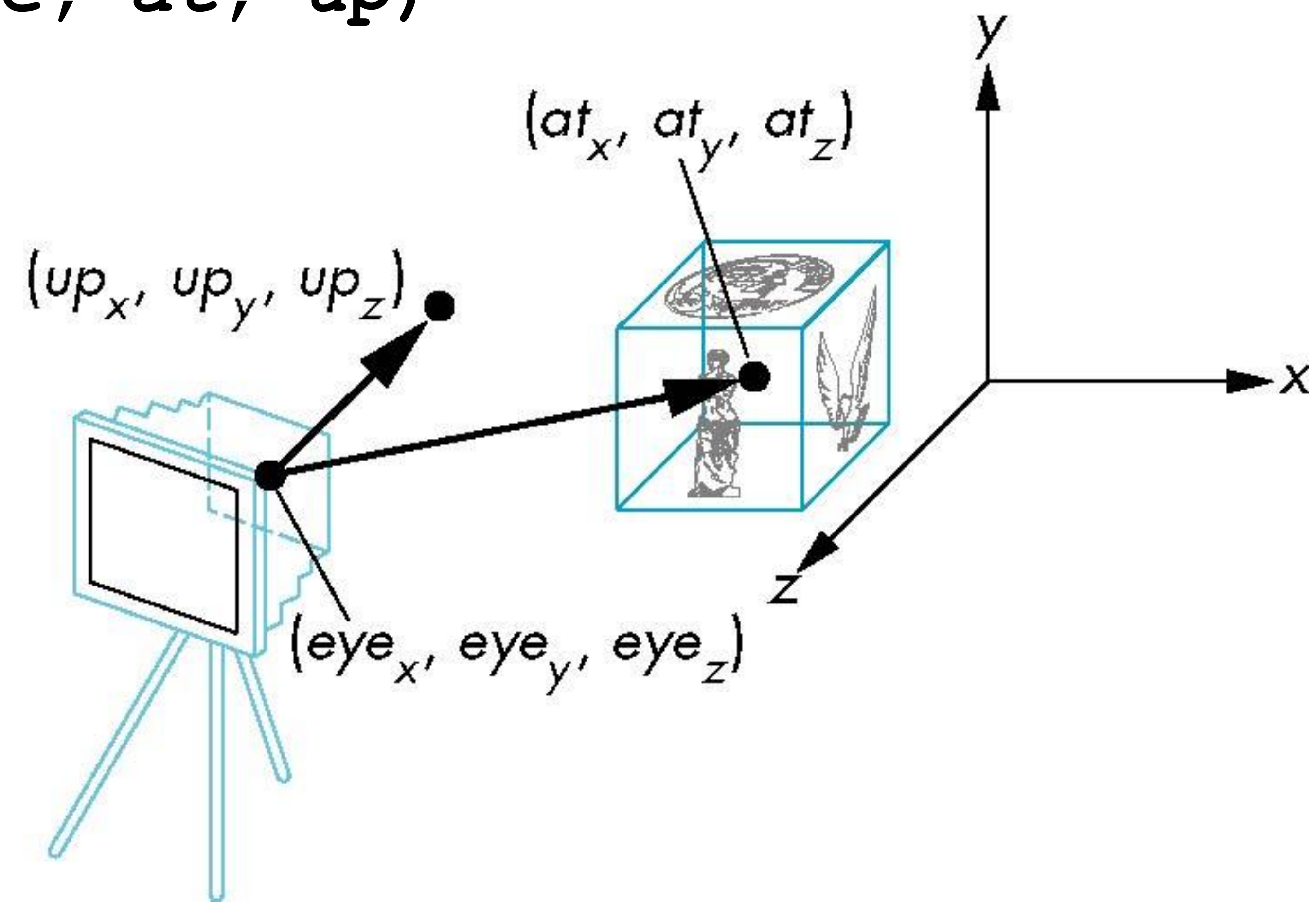
WebGL code

- Remember that last transformation specified is first to be applied

```
// Using MV.js

var t = translate (0.0, 0.0, -d);
var ry = rotateY(90.0);
var m = mult(t, ry);

or

var m = mult(translate (0.0, 0.0, -d),
        rotateY(90.0););
```

lookAt

```
LookAt(eye, at, up)
```



The lookAt Function

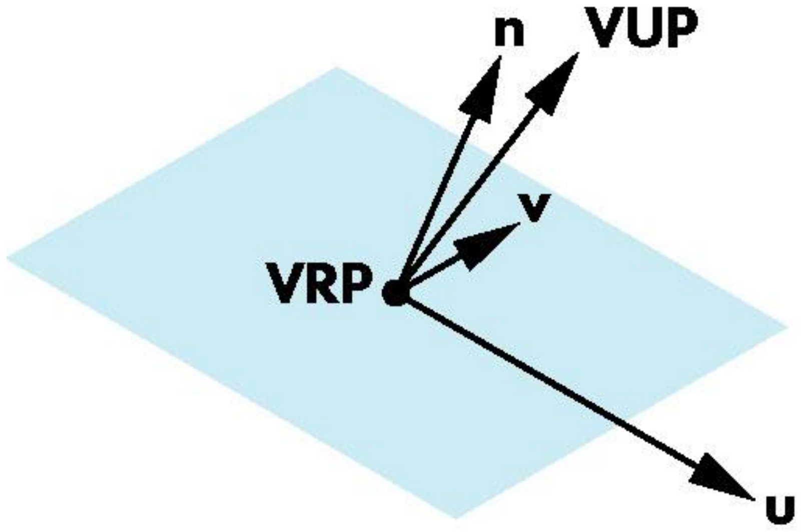
- The GLU library contained the function gluLookAt to form the required modelview matrix through a simple interface
- Note the need for setting an up direction
- Replaced by lookAt() in MV.js
  - Can concatenate with modeling transformations
- Example: isometric view of cube aligned with axes

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

var mv = LookAt(eye, at, up);
```

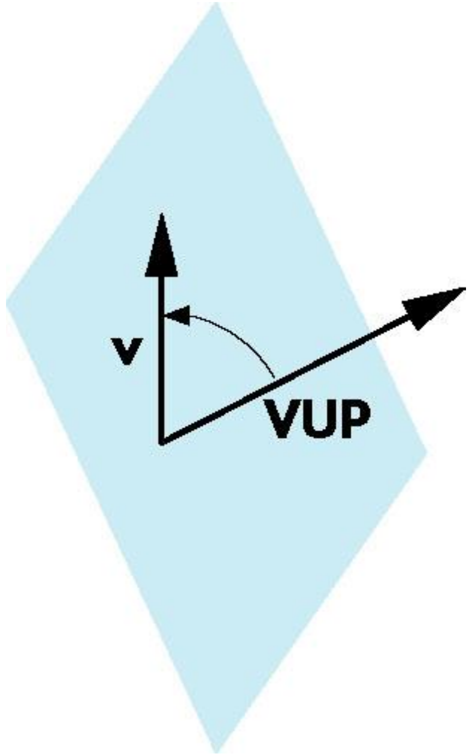
Other Viewing APIs

- The LookAt function is only one possible API for positioning the camera
- Others include
  - View reference point, view plane normal, view up (PHIGS, GKS-3D)
  - Yaw, pitch, roll
  - Elevation, azimuth, twist
  - Direction angles

View reference point, view plane normal, view up (PHIGS, GKS-3D)

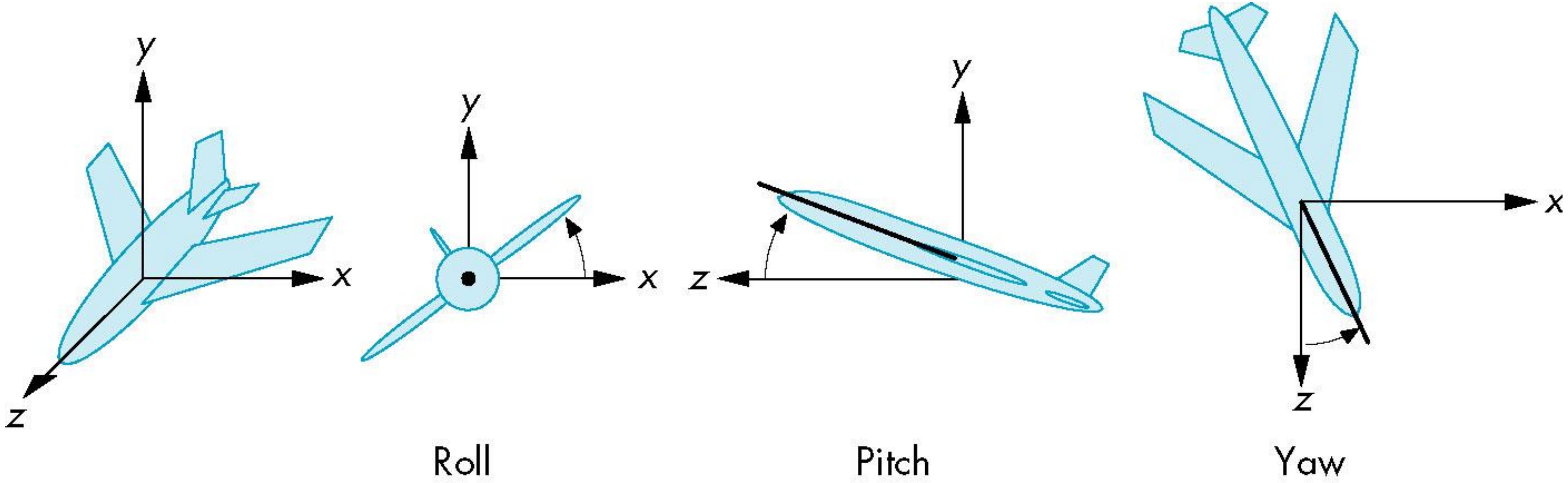


Camera frame

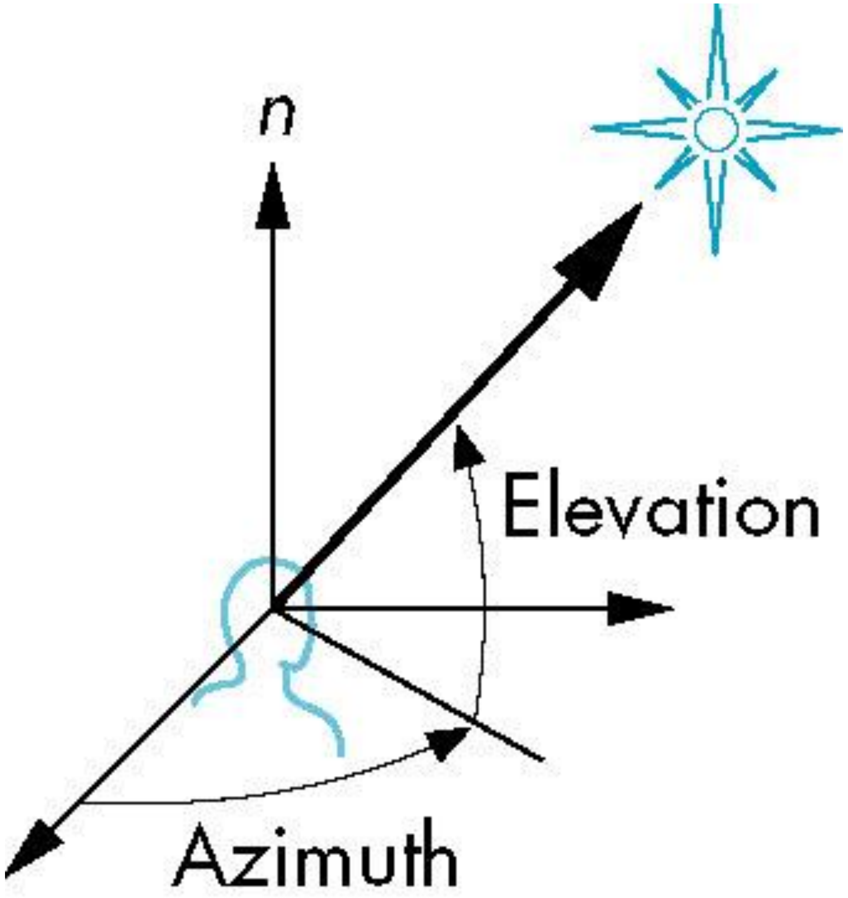


Determination of the view-up vector

Yaw, Pitch, and Roll



Elevation, Azimuth, and Twist



Computer Viewing
Projection

Objectives

- Introduce the mathematics of projection
- Add WebGL projection functions in MV.js

Projections and Normalization

- The default projection in the eye (camera) frame is orthogonal
- For points within the default view volume

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$

- Most graphics systems use *view normalization*
  - All other views are converted to the default view by transformations that determine the projection matrix
  - Allows use of the same pipeline for all views

Homogeneous Coordinate Representation

default orthographic projection

$x_p = x$
$y_p = y$
$z_p = 0$
$w_p = 1$

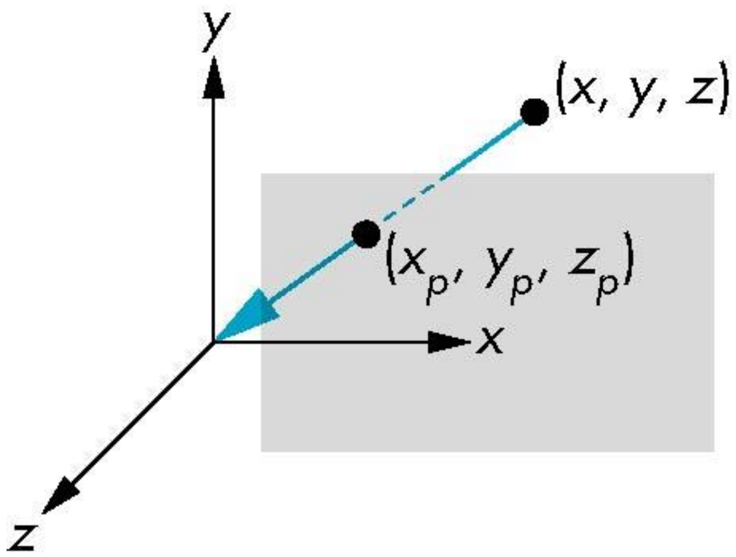
$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In practice, we can let $\mathbf{M} = \mathbf{I}$ and set the z term to zero later

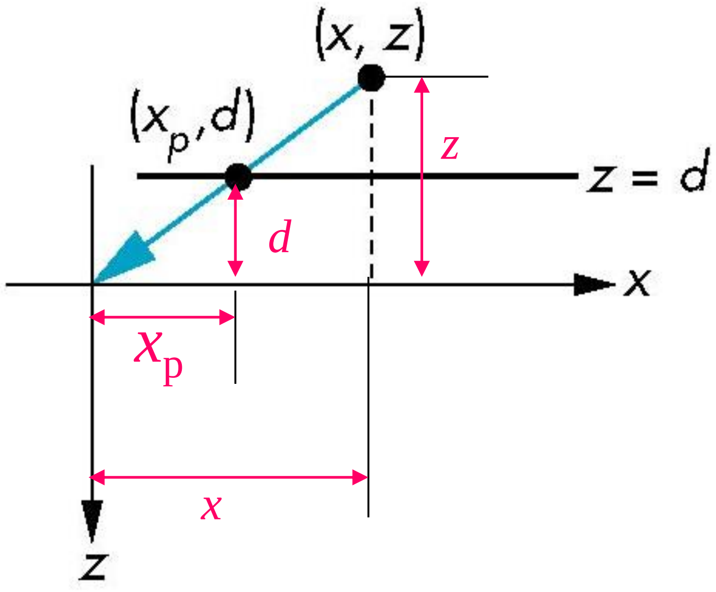
Simple Perspective

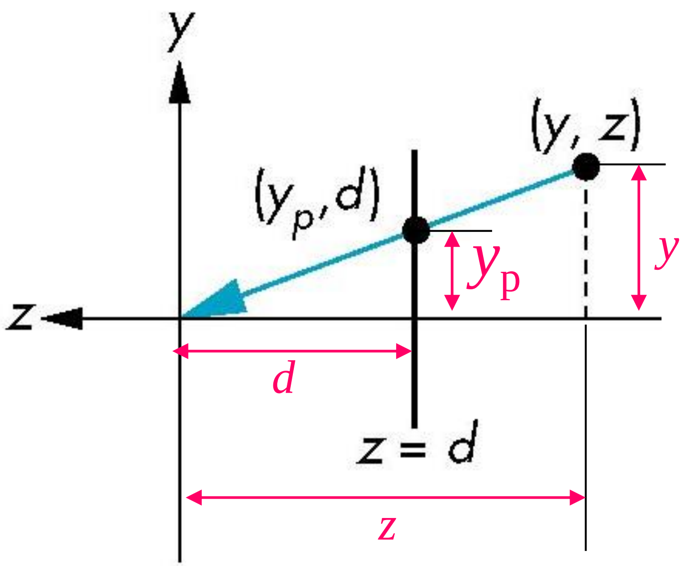
- Center of projection at the origin
- Projection plane $z = d$, $d < 0$



Perspective Equations

Consider top and side views





$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

Homogeneous Coordinate Form

consider $\mathbf{q} = \mathbf{Mp}$ where $\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

Perspective Division

- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates
- This *perspective division* yields

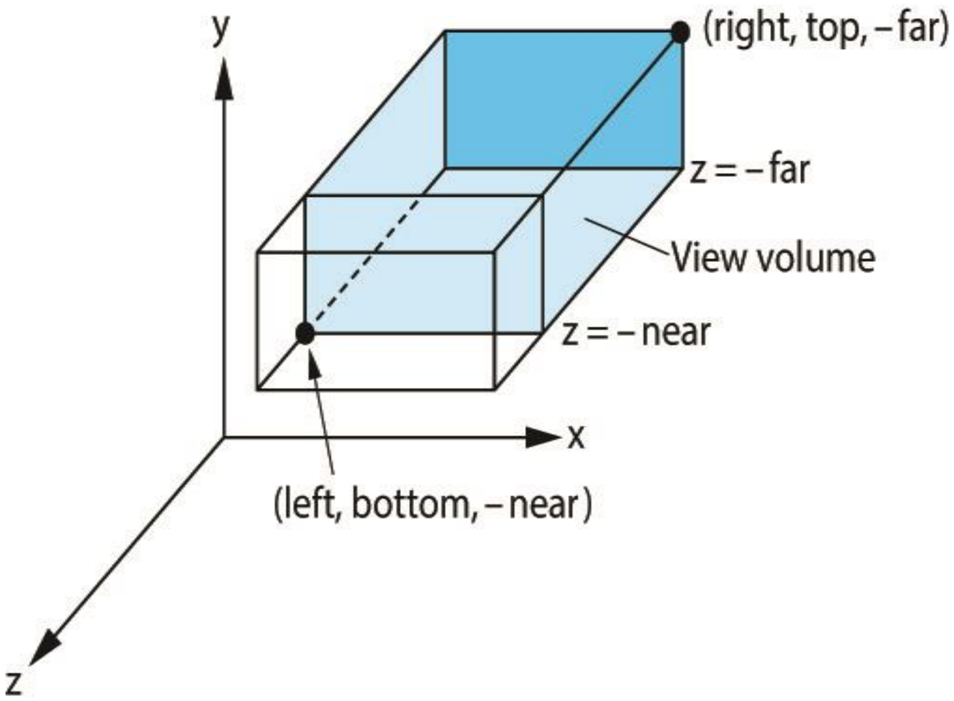
$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

the desired perspective equations

- We will consider the corresponding clipping volume with mat.h functions that are equivalent to deprecated OpenGL functions

WebGL Orthogonal Viewing

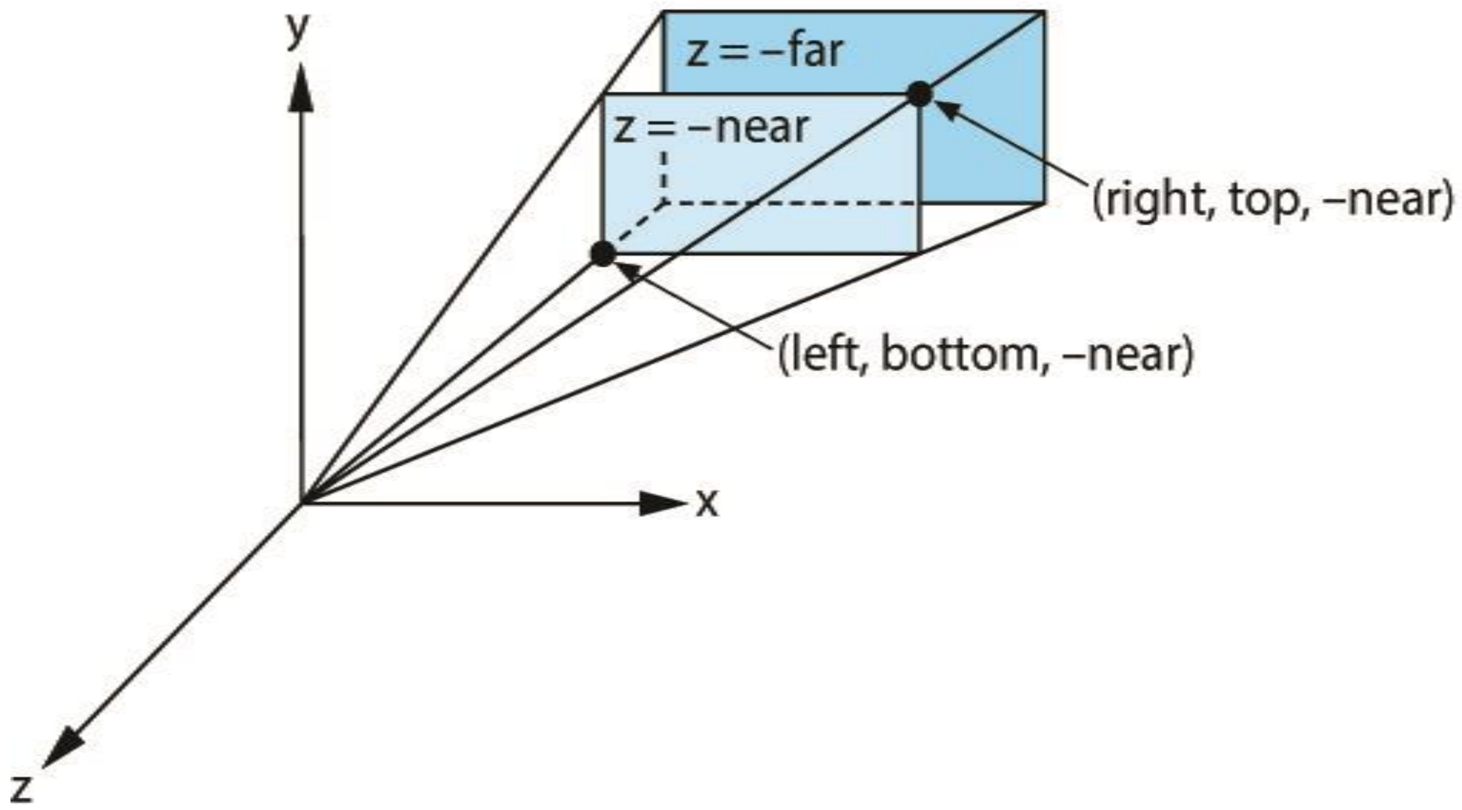
```
ortho(left,right,bottom,top,near,far)
```



`near` and `far` measured <u>from</u> camera

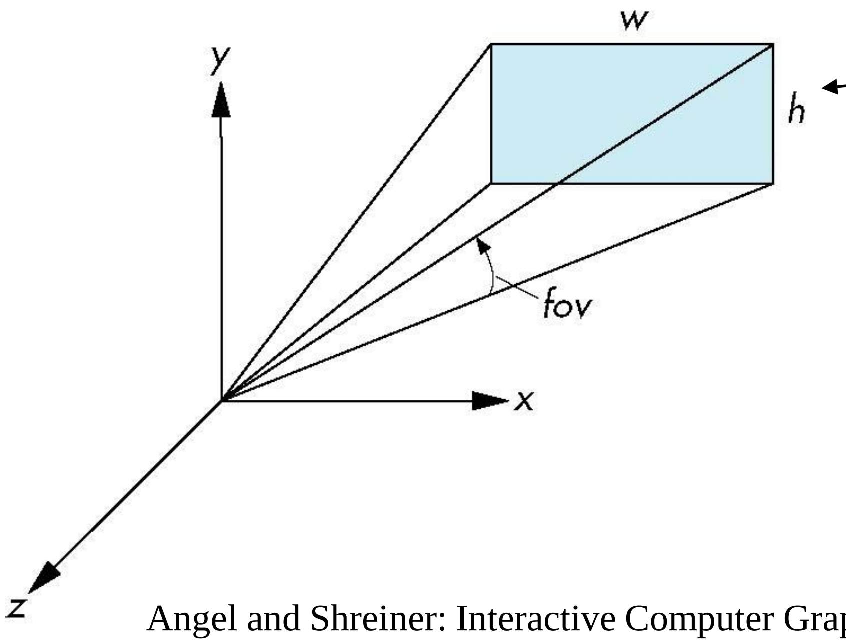
WebGL Perspective

`frustum(left,right,bottom,top,near,far)`



Using Field of View

- With **`frustum`** it is often difficult to get the desired view
- **`perpective(fovy, aspect, near, far)`** often provides a better interface

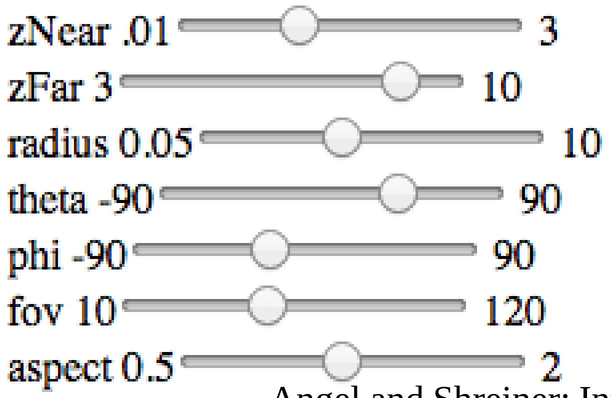


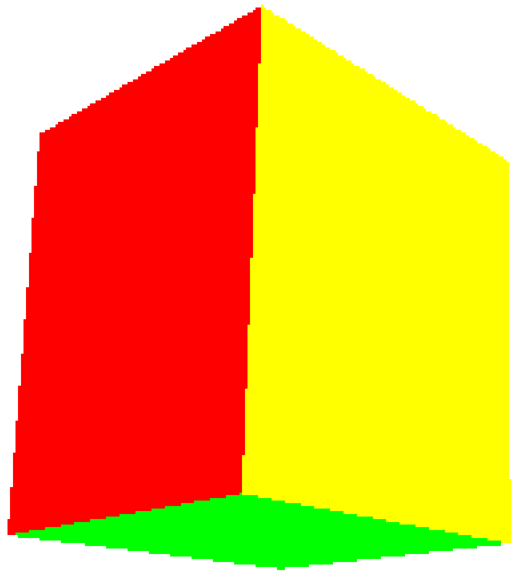
front plane

`aspect = w/h`

Computing Matrices

- Compute in JS file, send to vertex shader with gl.uniformMatrix4fv
- Dynamic: update in render() or shader





persepctive2.js

```
var render = function(){
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
                    radius*Math.sin(theta)*Math.sin(phi), radius*Math.cos(theta));
    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);
}
```

vertex shader

```
attribute  vec4 vPosition;
attribute  vec4 vColor;
varying vec4 fColor;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

void main() {
    gl_Position = projectionMatrix*modelViewMatrix*vPosition;
    fColor = vColor;
}
```

Orthogonal Projection Matrices

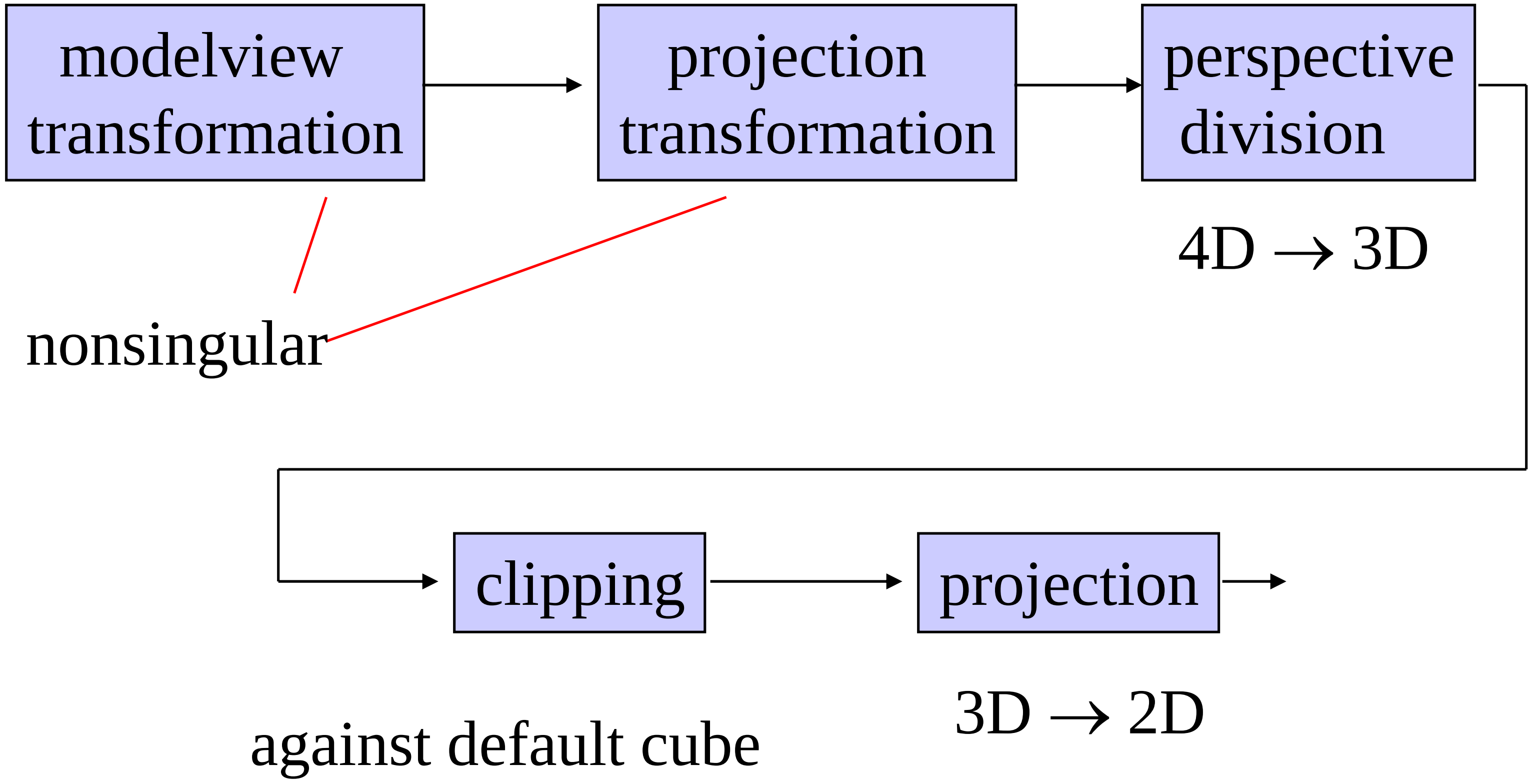
Objectives

- Derive the projection matrices used for standard orthogonal projections
- Introduce oblique projections
- Introduce projection normalization

Normalization

- Rather than derive a different projection matrix for each type of projection, we can convert all projections to orthogonal projections with the default view volume
- This strategy allows us to use standard transformations in the pipeline and makes for efficient clipping

Pipeline View



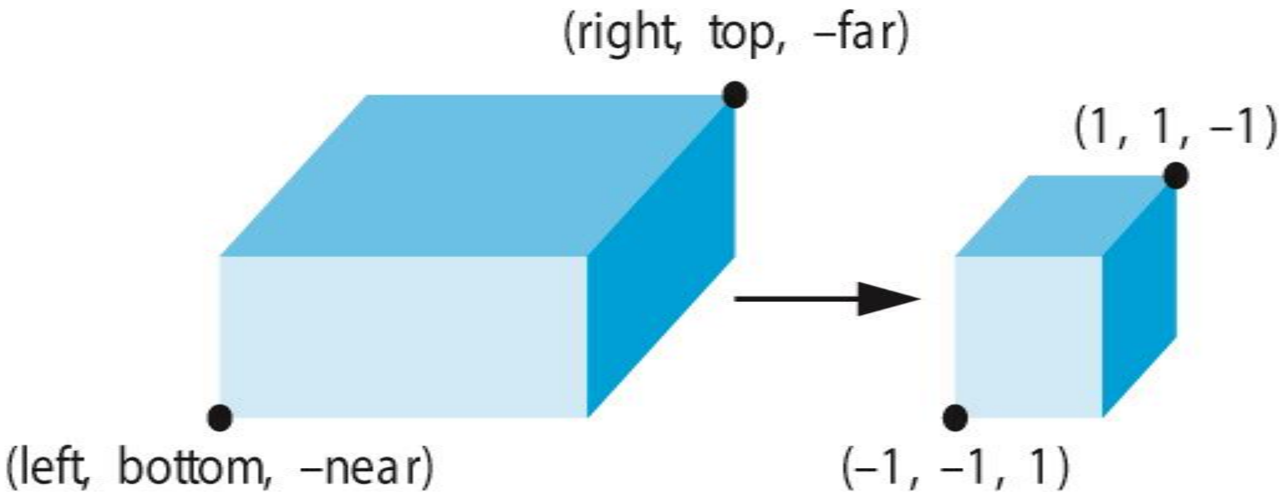
Notes

- We stay in four-dimensional homogeneous coordinates through both the modelview and projection transformations
  - Both these transformations are nonsingular
  - Default to identity matrices (orthogonal view)
- Normalization lets us clip against simple cube regardless of type of projection
- Delay final projection until end
  - Important for hidden-surface removal to retain depth information as long as possible

Orthogonal Normalization

```
ortho(left,right,bottom,top,near,far)
```

normalization ⇒ find transformation to convert specified clipping volume to default



Orthogonal Matrix

- Two steps
  - Move center to origin

    T(-(left+right)/2, -(bottom+top)/2,(near+far)/2))
  - Scale to have sides of length 2

    S(2/(left-right),2/(top-bottom),2/(near-far))

$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right-left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Final Projection

- Set $z = 0$
- Equivalent to the homogeneous coordinate transformation

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Hence, general orthogonal projection in 4D is

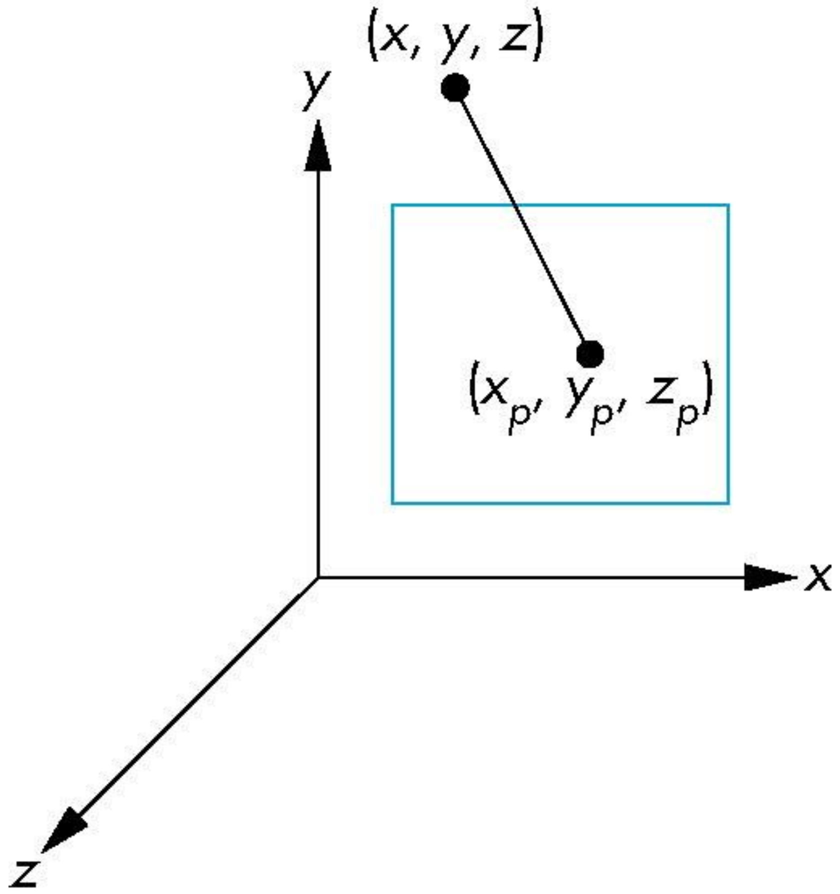
$$\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{ST}$$

Oblique Projections

- The OpenGL projection functions cannot produce general parallel projections such as

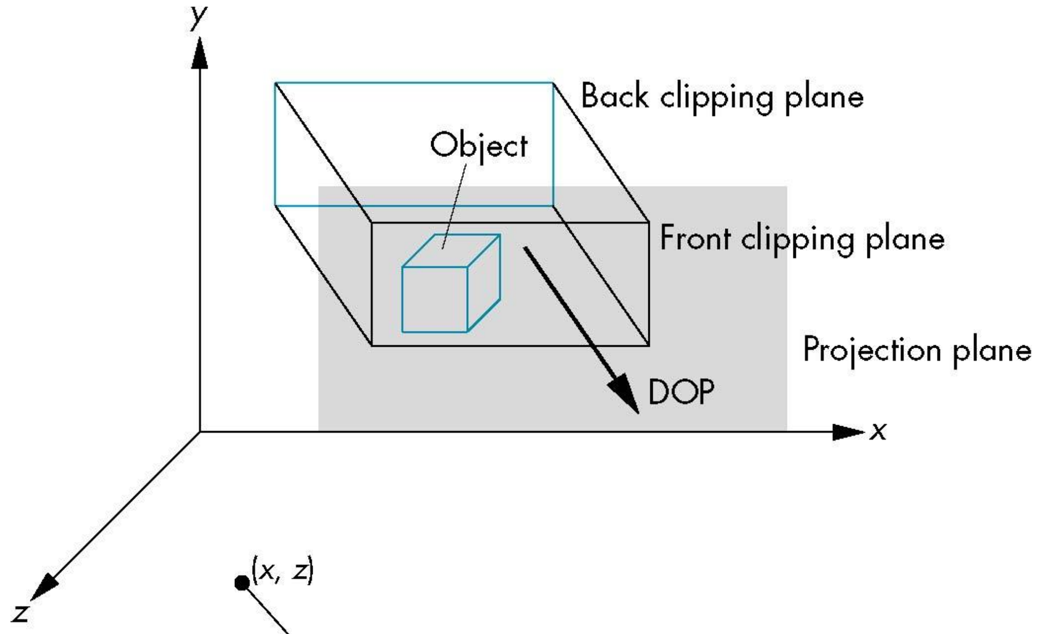
- However if we look at the example of the cube it appears that the cube has been sheared
- Oblique Projection = Shear + Orthogonal Projection

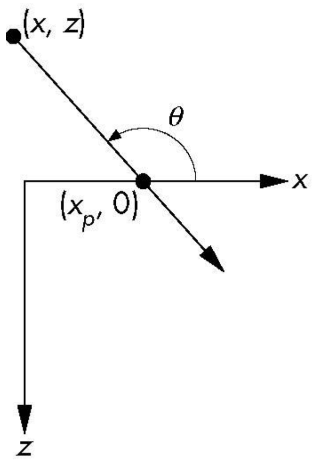
Oblique Projections



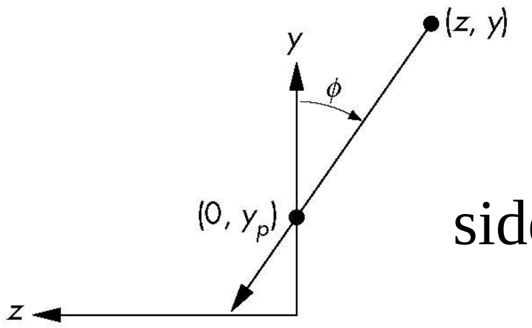
General Shear

Oblique Projection





top view



side view

Shear Matrix

xy shear (*z* values unchanged)

$$\mathbf{H}(\theta,\phi) = \begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\varphi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

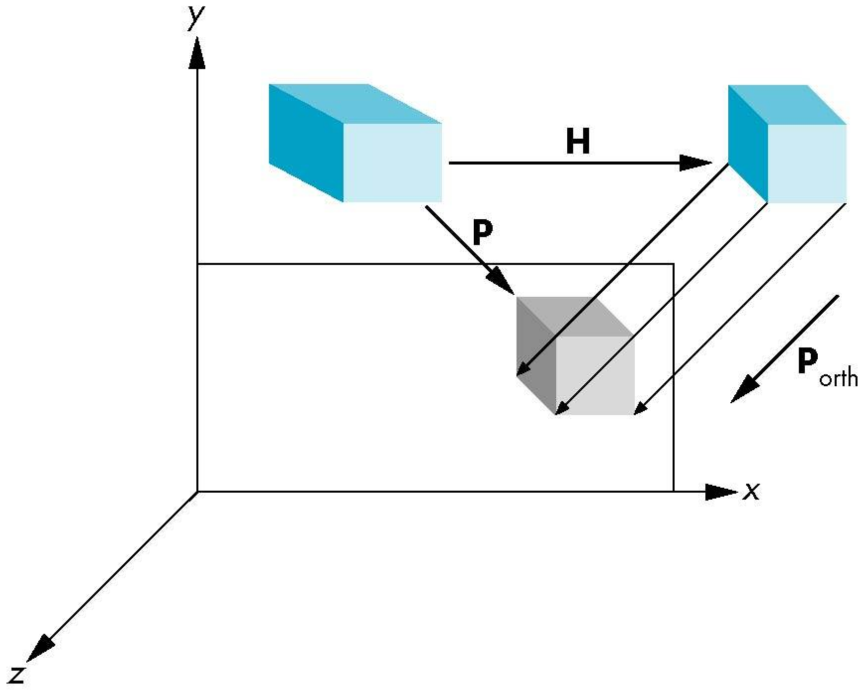
Projection matrix

$$\mathbf{P} = \mathbf{M}_{\text{orth}}\,\mathbf{H}(\theta,\phi)$$

General case:

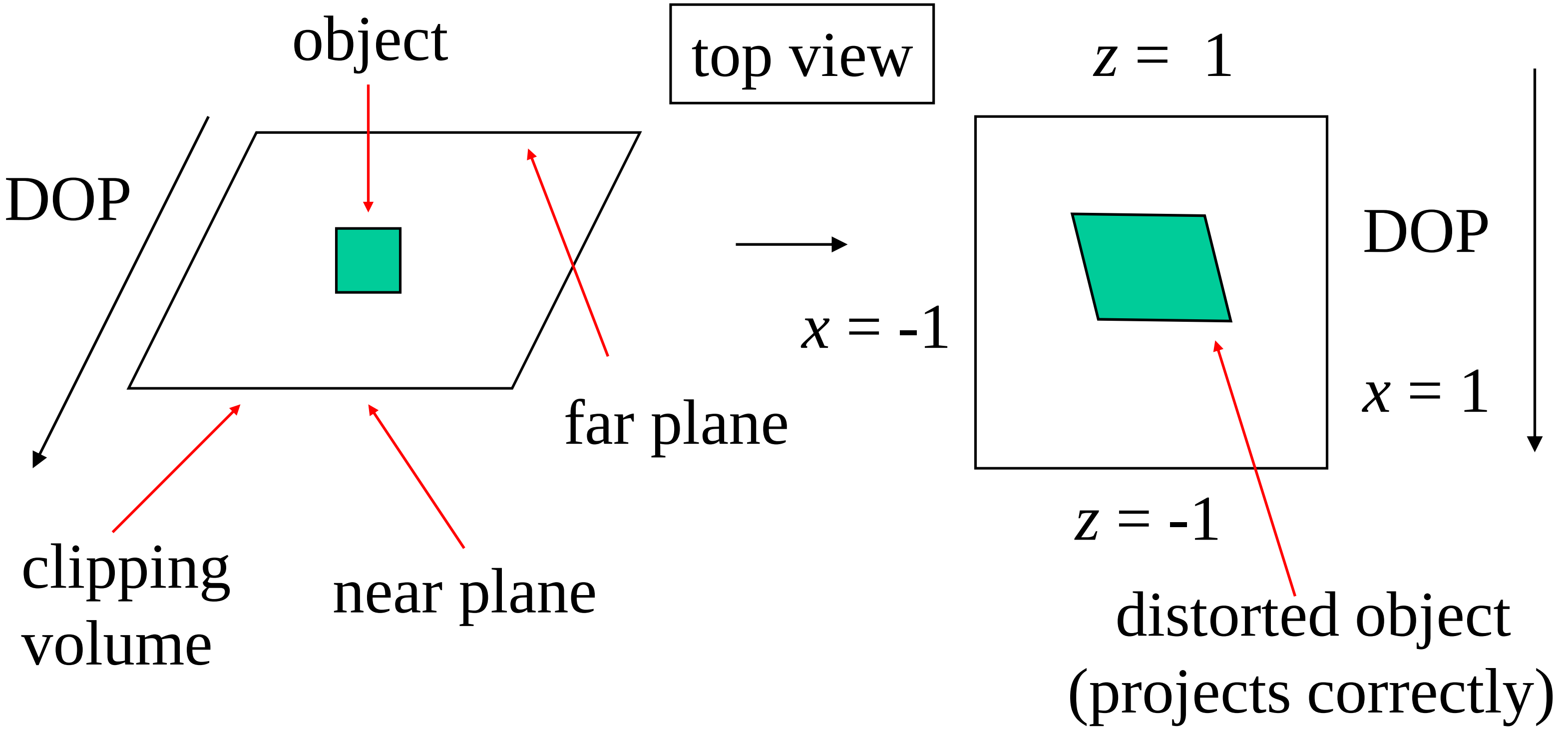
$$\mathbf{P} = \mathbf{M}_{\text{orth}}\,\mathbf{STH}(\theta,\phi)$$

Equivalency



Effect on Clipping

- The projection matrix **P** = **STH** transforms the original clipping volume to the default clipping volume



Perspective Projection Matrices

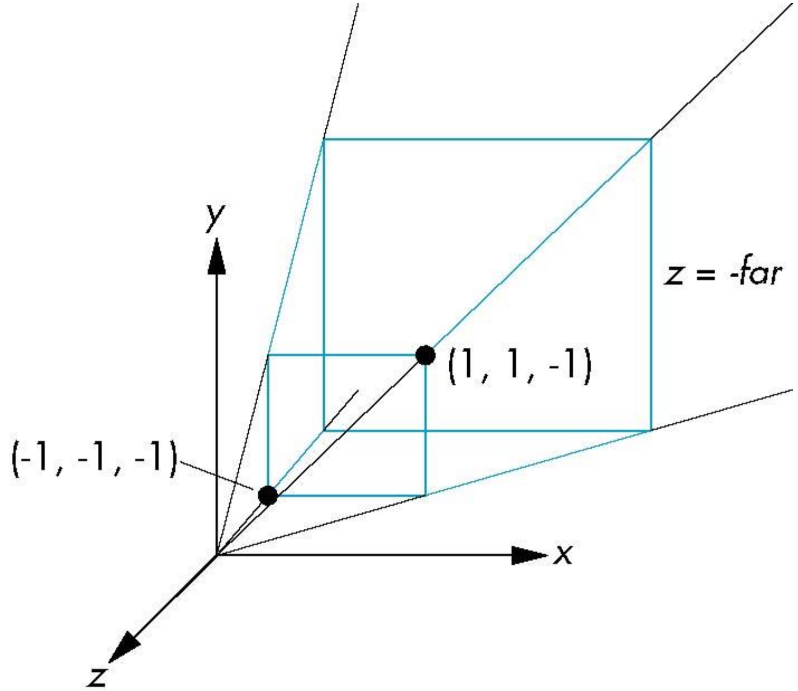
Objectives

- Derive the perspective projection matrices used for standard WebGL projections

Simple Perspective

Consider a simple perspective with the COP at the origin, the near clipping plane at z = -1, and a 90 degree field of view determined by the planes

$x = \pm z,\ y = \pm z$



Perspective Matrices

Simple projection matrix in homogeneous coordinates

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Note that this matrix is independent of the far clipping plane

Generalization

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

after perspective division, the point $(x, y, z, 1)$ goes to

$x'' = x/z$
$y'' = y/z$
$Z'' = -(\alpha+\beta/z)$

which projects orthogonally to the desired point regardless of α and β

Picking α and β

If we pick

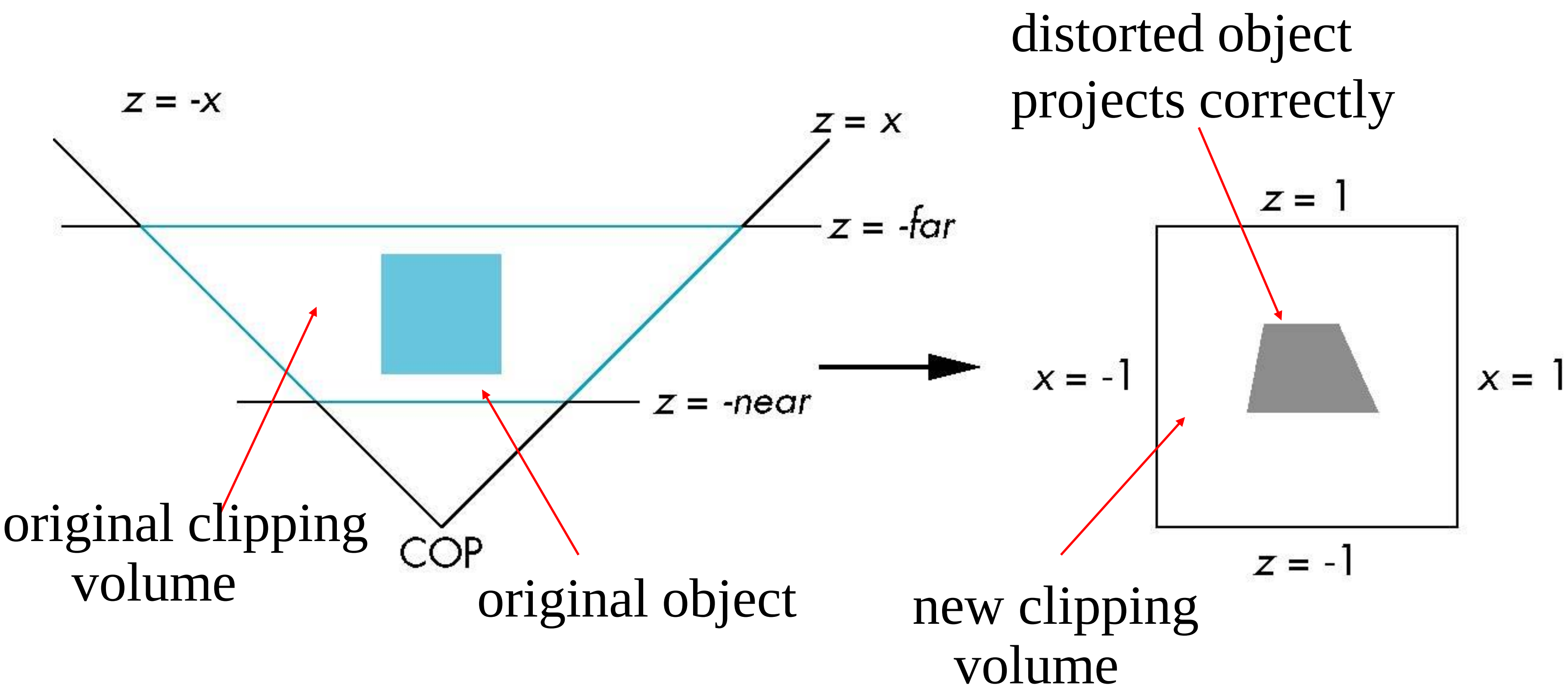
$$\alpha = \frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$

$$\beta = \frac{2\text{near} * \text{far}}{\text{near} - \text{far}}$$

the near plane is mapped to $z = -1$
the far plane is mapped to $z = 1$
and the sides are mapped to $x = \pm 1$, $y = \pm 1$

Hence the new clipping volume is the default clipping volume

Normalization Transformation

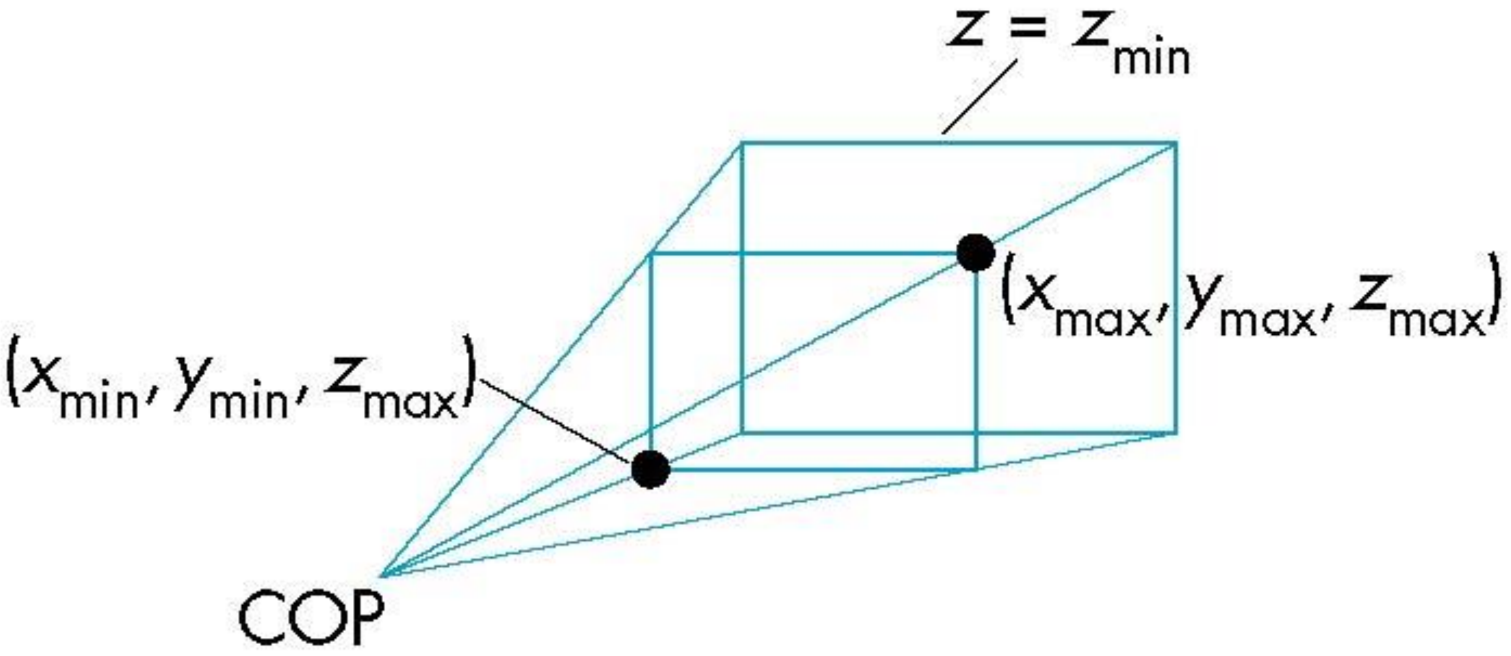


Normalization and Hidden-Surface Removal

- Although our selection of the form of the perspective matrices may appear somewhat arbitrary, it was chosen so that if $z_1 > z_2$ in the original clipping volume then the for the transformed points $z_1' > z_2'$
- Thus hidden surface removal works if we first apply the normalization transformation
- However, the formula $z'' = -(\alpha+\beta/z)$ implies that the distances are distorted by the normalization which can cause numerical problems especially if the near distance is small

WebGL Perspective

- `gl.frustum` allows for an unsymmetric viewing frustum (although `gl.perspective` does not)



OpenGL Perspective Matrix

- The normalization in `Frustum` requires an initial shear to form a right viewing pyramid, followed by a scaling to get the normalized perspective volume. Finally, the perspective matrix results in needing only a final orthogonal transformation

$$\mathbf{P} = \mathbf{NSH}$$

our previously defined perspective matrix

shear and scale

Why do we do it this way?

- Normalization allows for a single pipeline for both perspective and orthogonal viewing
- We stay in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading
- We simplify clipping

Perspective Matrices

frustum

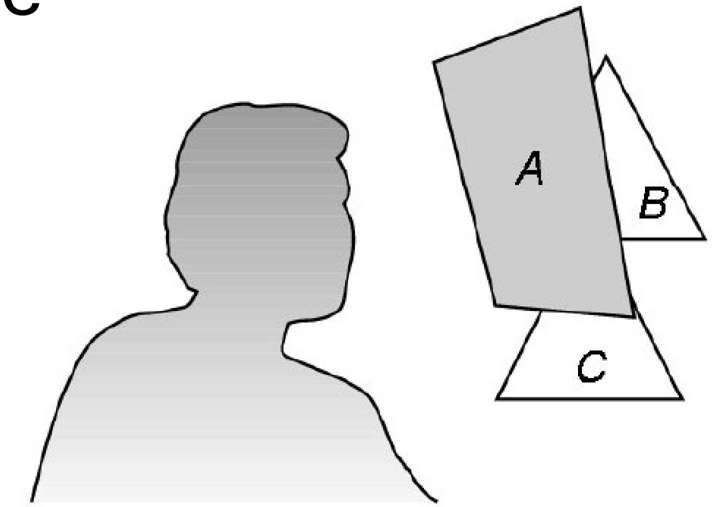
$$\mathbf{P} = \begin{bmatrix} \frac{2*near}{right-left} & 0 & \frac{right-left}{right-left} & 0 \\ 0 & \frac{2*near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2*far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

perspective

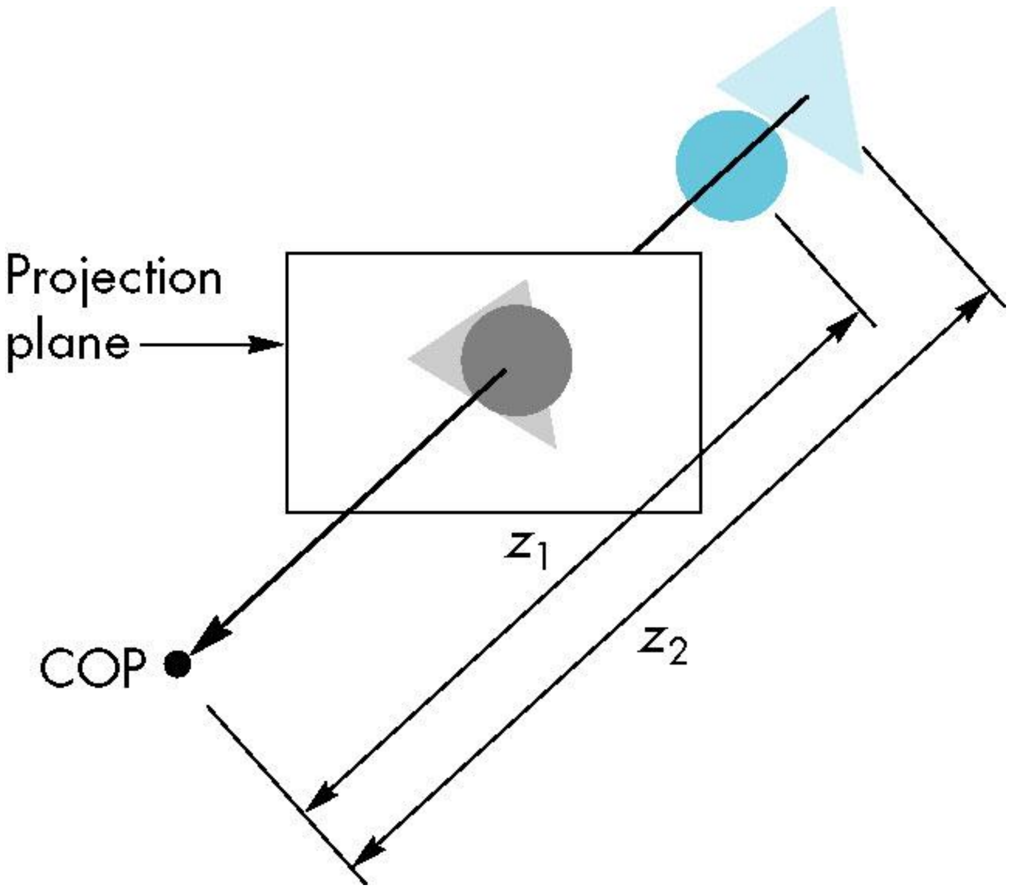
$$\mathbf{P} = \begin{bmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2*far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Hidden-Surface Removal

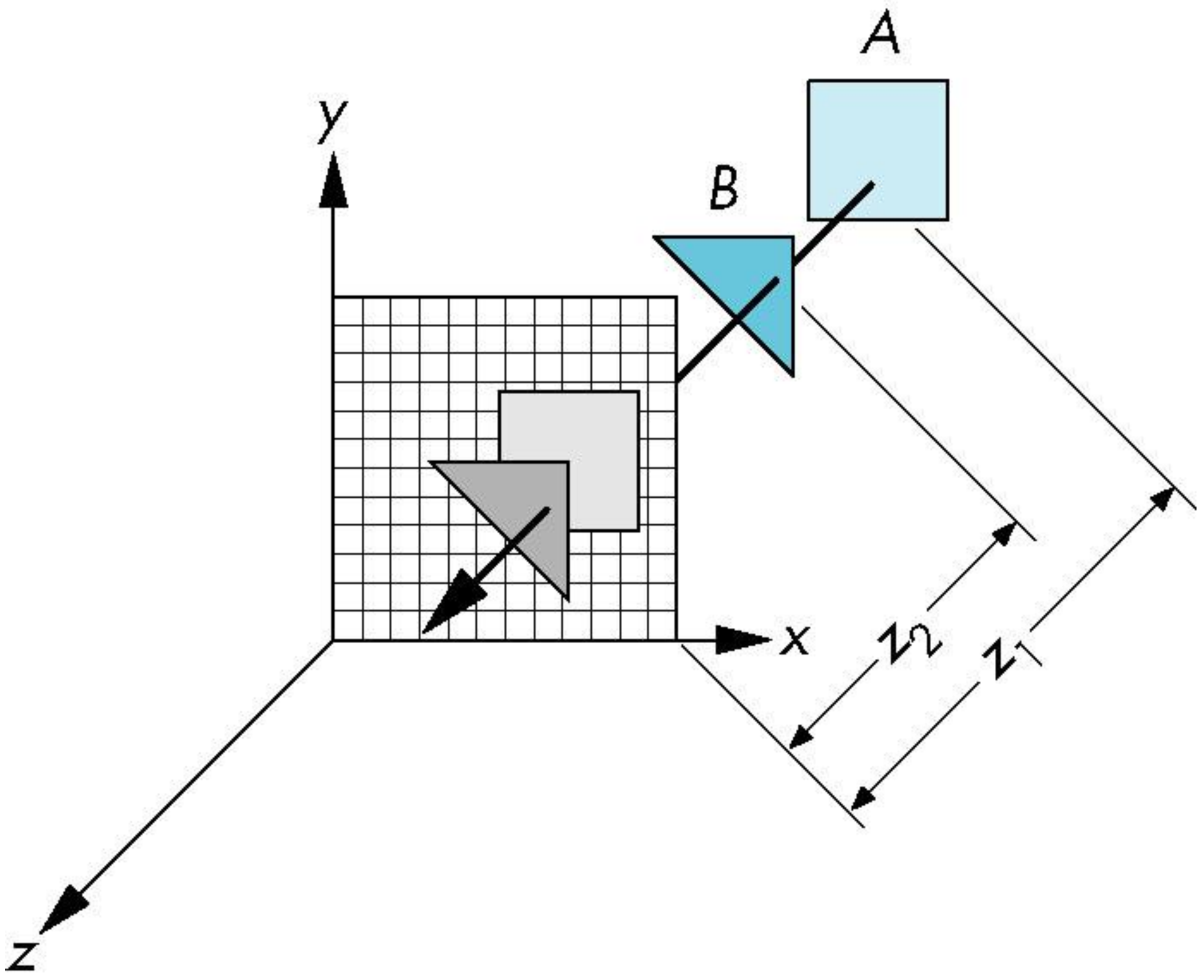
- We want to see only those surfaces in front of other surfaces
- OpenGL uses a hidden-surface method called the z-buffer algorithm that saves depth information as objects are rendered so that only the front objects appear in the image



The Z-buffer Algorithm



The Z-buffer Algorithm



Meshes

Objective

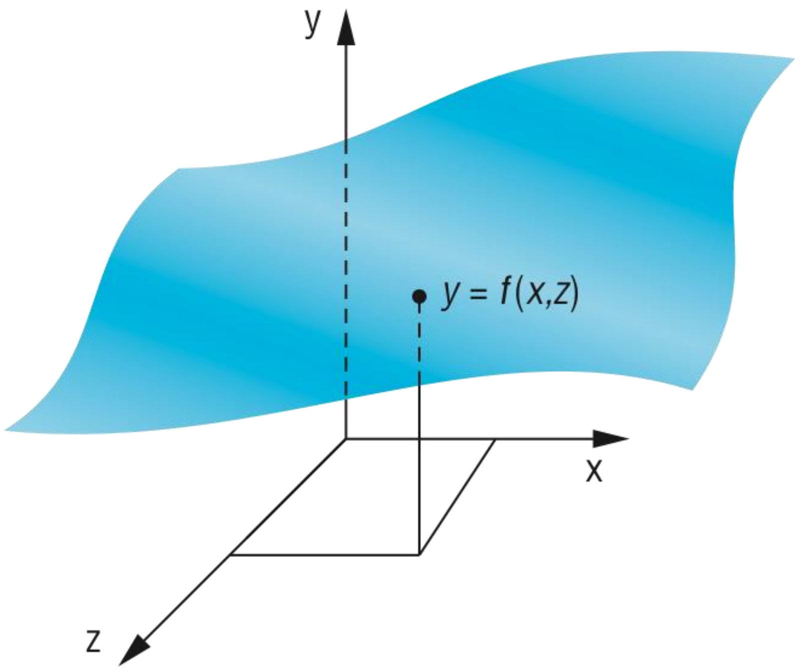
Introduce techniques for displaying polygonal meshes

Meshes

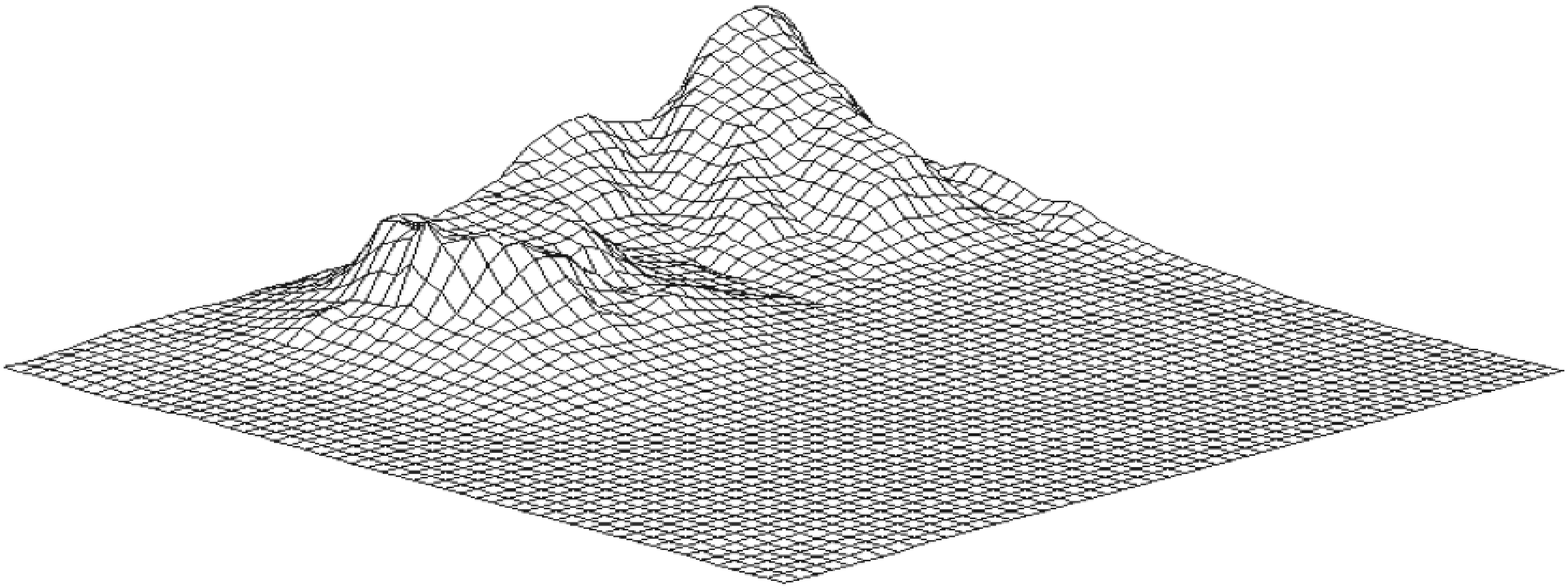
- Polygonal meshes are the standard method for defining and displaying surfaces
  - Approximations to curved surfaces
  - Directly from CAD packages
  - Subdivision
- Most common are quadrilateral and triangular meshes
  - Triangle strips and fans

Height Fields

- For each (x, z) there is a unique y
- Sampling leads to an array of y values
- Display as quadrilateral or triangular mesh using strips



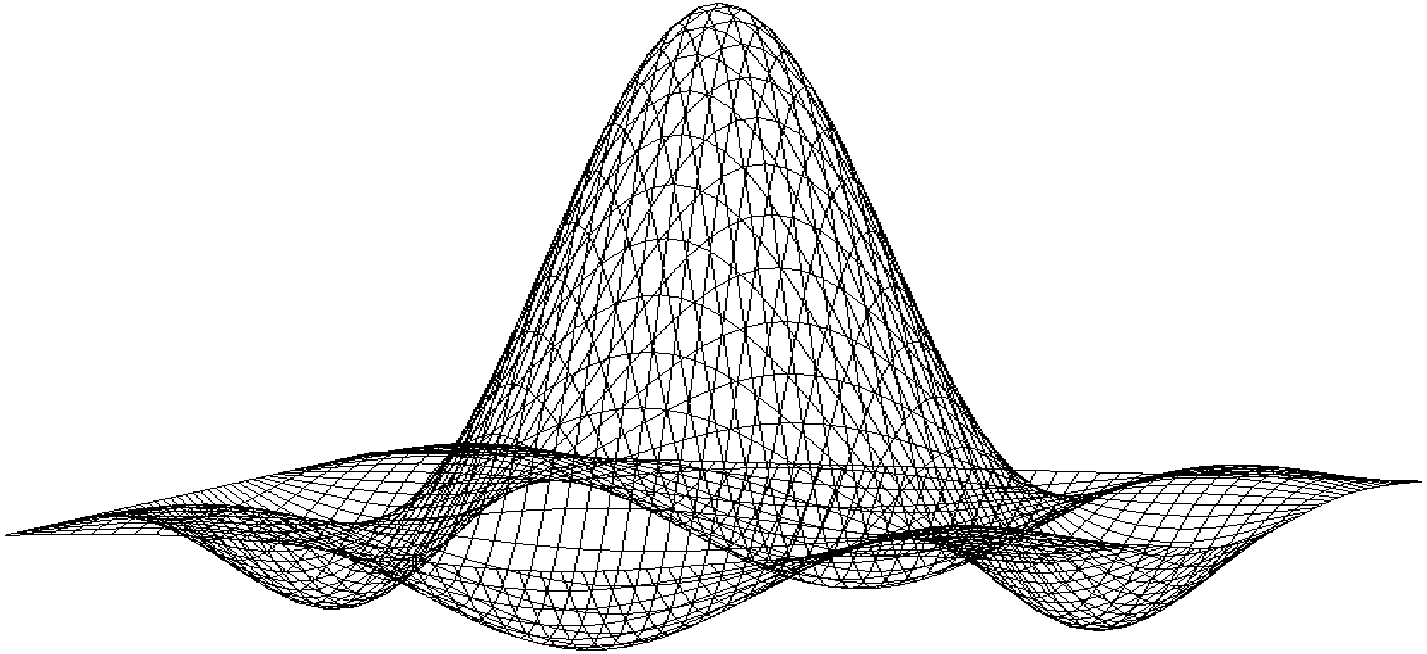
Honolulu Plot Using Line Strips



Plot 3D

- Old 2D method uses fact that data are ordered and we can render front to back
- Regard each plane of constant z as a flat surface that can block (parts of) planes behind it
- Can proceed iteratively maintaining a visible top and visible bottom
  - Lots of little line intersections
- Lots of code but avoids all 3D

Lines on Back and Hidden Faces

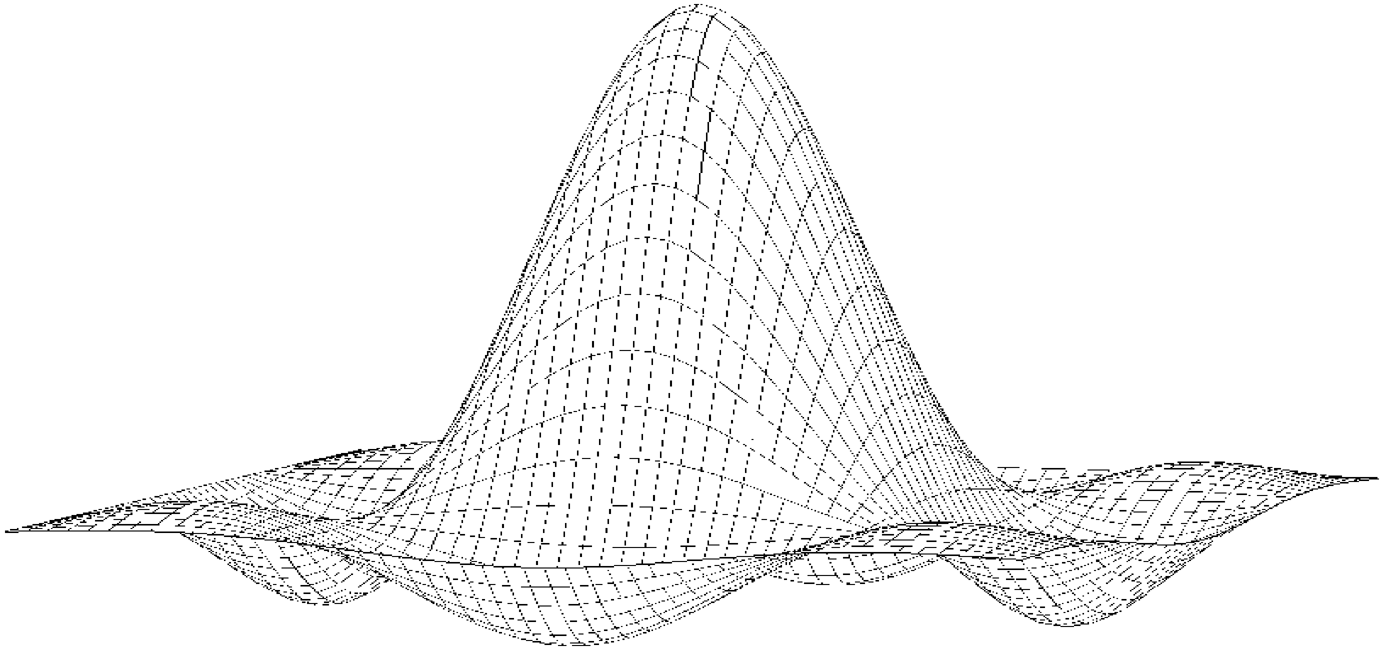


sombrero or Mexican hat function $(\sin \pi r)/(\pi r)$

Using Polygons

- We can use four adjacent data points to form a quadrilateral and thus two triangles which can be shaded
- But what if we want to see the grid?
- We can display each quadrilateral twice
    - First as two filled white triangles
    - Second as a black line loop

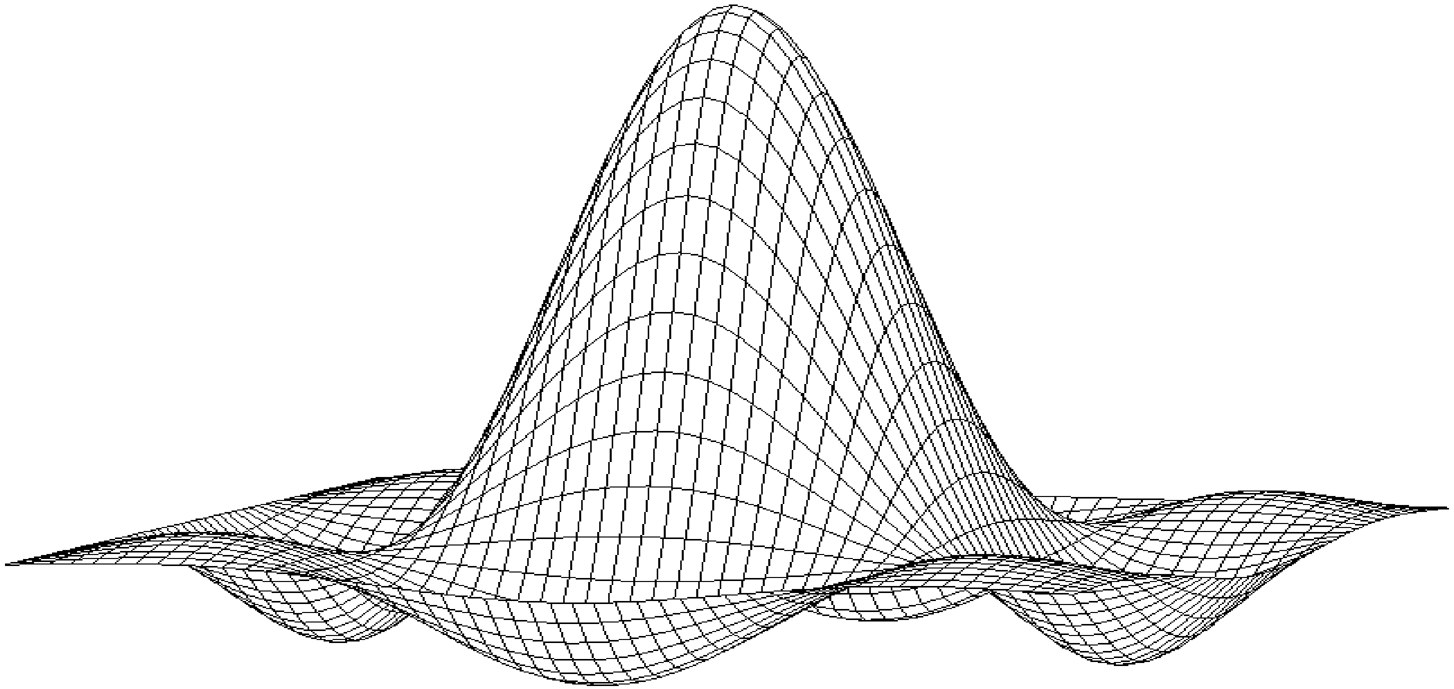
Hat Using Triangles and Lines



Polygon Offset

- Even though we draw the polygon first followed by the lines, small numerical errors cause some of fragments on the line to be display behind the corresponding fragment on the triangle
- Polygon offset (gl.polygonOffset) moves fragments slight away from camera
- Apply to triangle rendering

Hat with Polygon Offset



Other Mesh Issues

- How do we construct a mesh from disparate data (unstructured points)
- Technologies such as laser scans can produced tens of millions of such points
- Chapter 12: Delaunay triangulation
- Can we use one triangle strip for an entire 2D mesh?
- Mesh simplification

Shadows

Objectives

- Introduce Shadow Algorithms
- Projective Shadows
- Shadow Maps
- Shadow Volumes

Flashlight in the Eye Graphics

- When do we not see shadows in a real scene?
- When the only light source is a point source at the eye or center of projection
  - Shadows are behind objects and not visible
- Shadows are a global rendering issue
  - Is a surface visible from a source
  - May be obscured by other objects

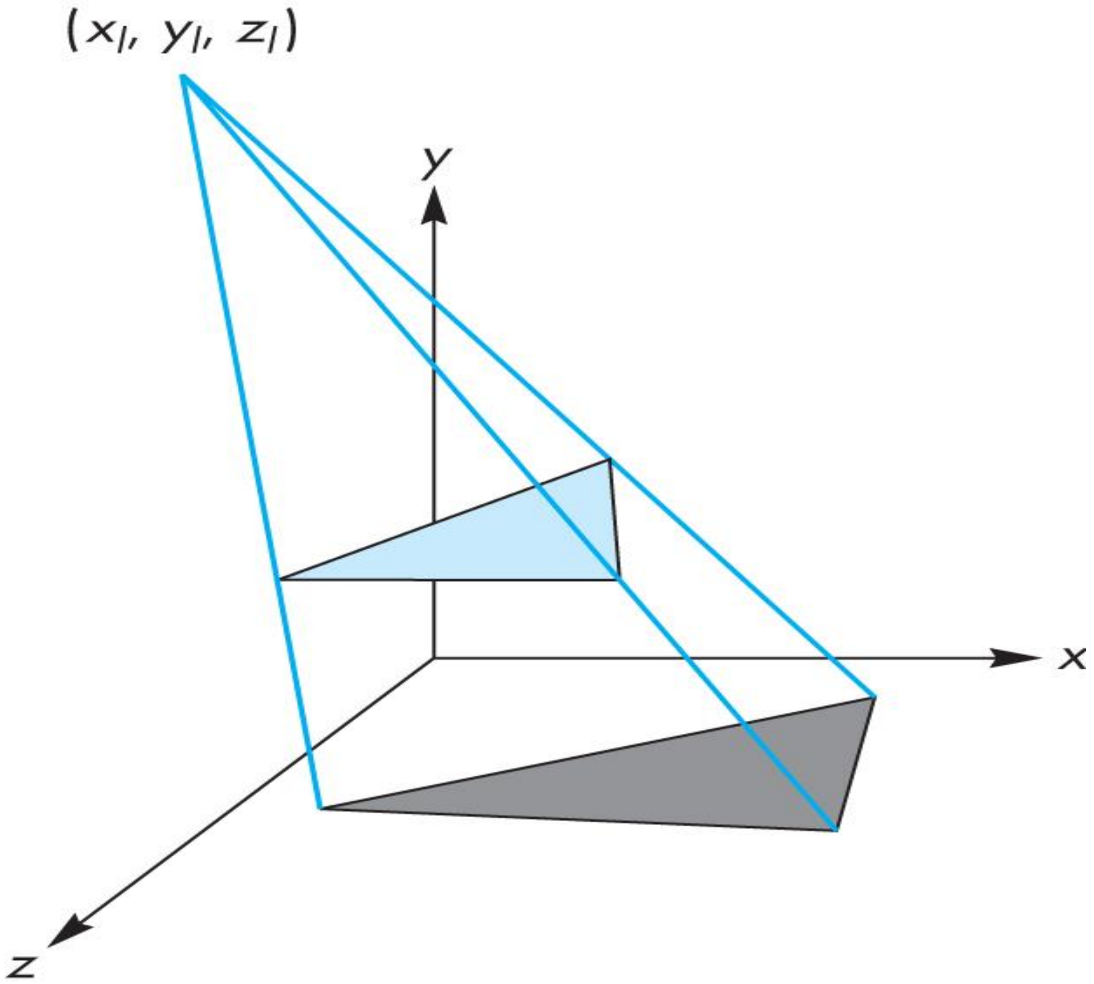
Shadows in Pipeline Renders

- Note that shadows are generated automatically by a ray tracers
  - feeler rays will detect if no light reaches a point
  - need all objects to be available
- Pipeline renderers work an object at a time so shadows are not automatic
  - can use some tricks: projective shadows
  - multi-rendering: shadow maps and shadow volumes

Projective Shadows

- Oldest methods
  - Used in flight simulators to provide visual clues
- Projection of a polygon is a polygon called a **shadow polygon**
- Given a point light source and a polygon, the vertices of the shadow polygon are the projections of the original polygon’s vertices from a point source onto a surface

Shadow Polygon



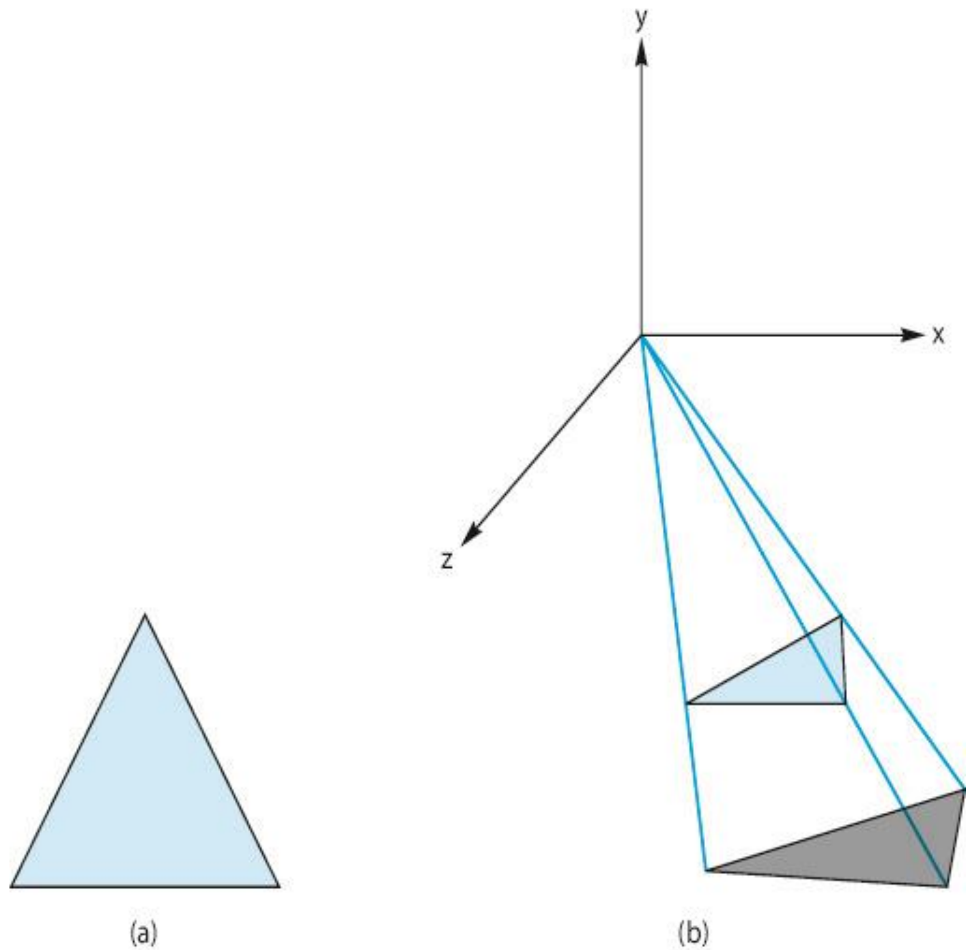
Computing Shadow Vertex

1. Source at (x_l, y_l, z_l)
2. Vertex at (x, y, z)
3. Consider simple case of shadow projected onto ground at (x_p, 0, z_p)
4. Translate source to origin with T($-x_l$, $-y_l$, $-z_l$)
5. Perspective projection

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{-y_l} & 0 & 0 \end{bmatrix}$$

6. Translate back

Shadow Polygon Projection



From a light source

With source moved to the origin

Matrix Operation

modelViewMatrix=$T(x_l, y_l, z_l) \cdot M \cdot T(-x_l, -y_l, -z_l)$

$$\text{where } M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{-1}{y_l} & 0 & 0 \end{bmatrix}$$

Light source $L(x_l, y_l, z_l, 1)$

Point $P(x, y, z, 1)$

Projection point $P'(x_p, y_p, z_p, 1)$ (on y=0 plane)

$$\boxed{P'^T = \text{modelViewMatrix} \cdot P^T}$$

$$\begin{cases} x_p = x_l - \dfrac{x - x_l}{(y - y_l)/y_l} \\ y_p = 0 \\ z_p = z_l - \dfrac{z - z_l}{(y - y_l)/y_l} \end{cases}$$

Shadow Process

1. Put two identical triangles and their colors on GPU (black for shadow triangle)
2. Compute two model view matrices as uniforms
3. Send model view matrix for original triangle
4. Render original triangle
5. Send second model view matrix
6. Render shadow triangle

- Note shadow triangle undergoes two transformations
- Note hidden surface removal takes care of depth issues

Generalized Shadows

- Approach was OK for shadows on a single flat surface
- Note with geometry shader we can have the shader create the second triangle
- Cannot handle shadows on general objects
- Exist a variety of other methods based on same basic idea
- We'll pursue methods based on projective textures

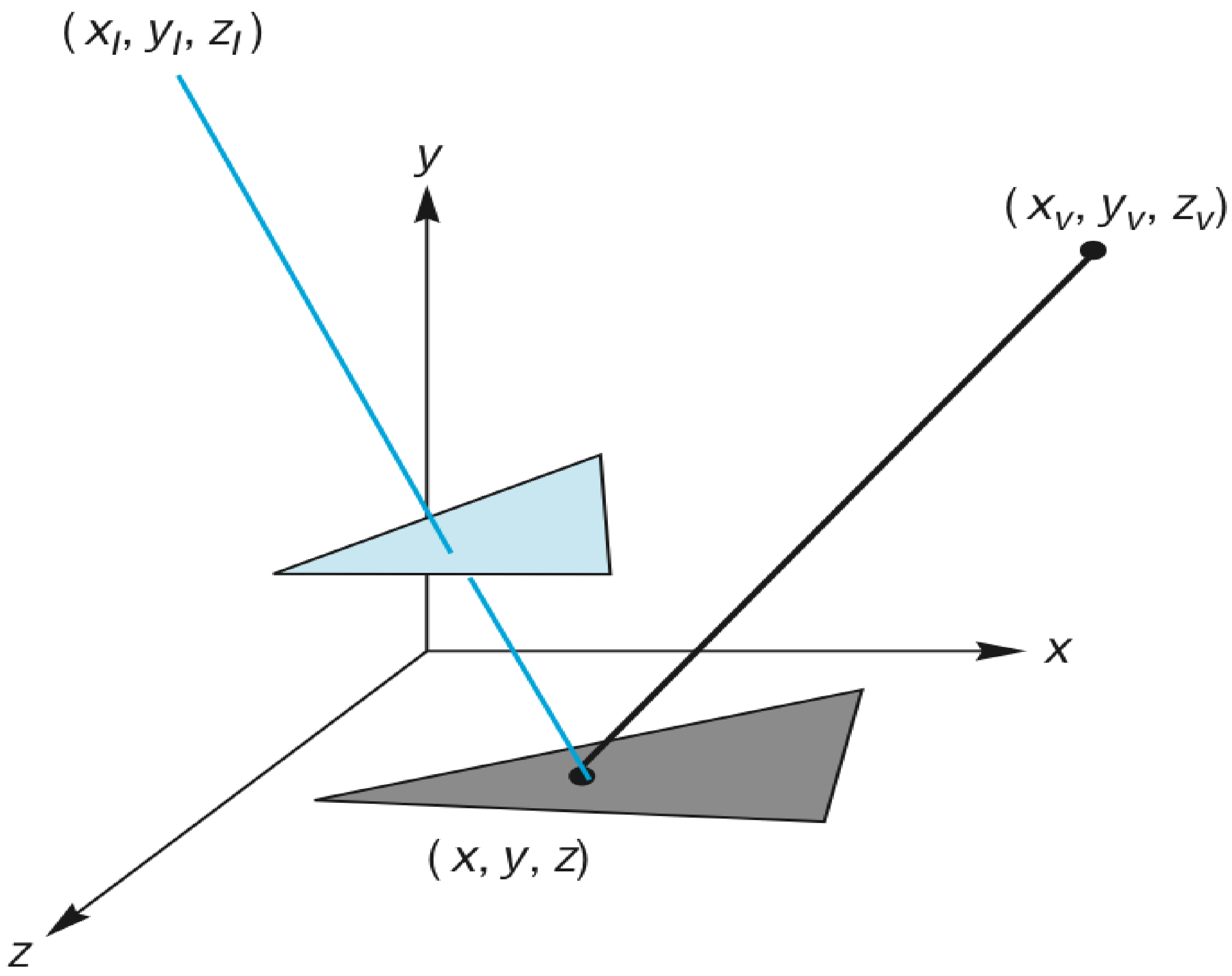
Image Based Lighting

- We can project a texture onto the surface in which case the are treating the texture as a “slide projector”
- This technique the basis of projective textures and image based lighting
- Supported in desktop OpenGL and GLSL through four dimensional texture coordinates
- Not yet in WebGL

Shadow Maps

- If we render a scene from a light source, the depth buffer will contain the distances from the source to nearest lit fragment.
- We can store these depths in a texture called a **depth map** or **shadow map**
- Note that although we don't care about the image in the shadow map, if we render with some light, anything lit is not in shadow.
- Form a shadow map for each source

Shadow Mapping



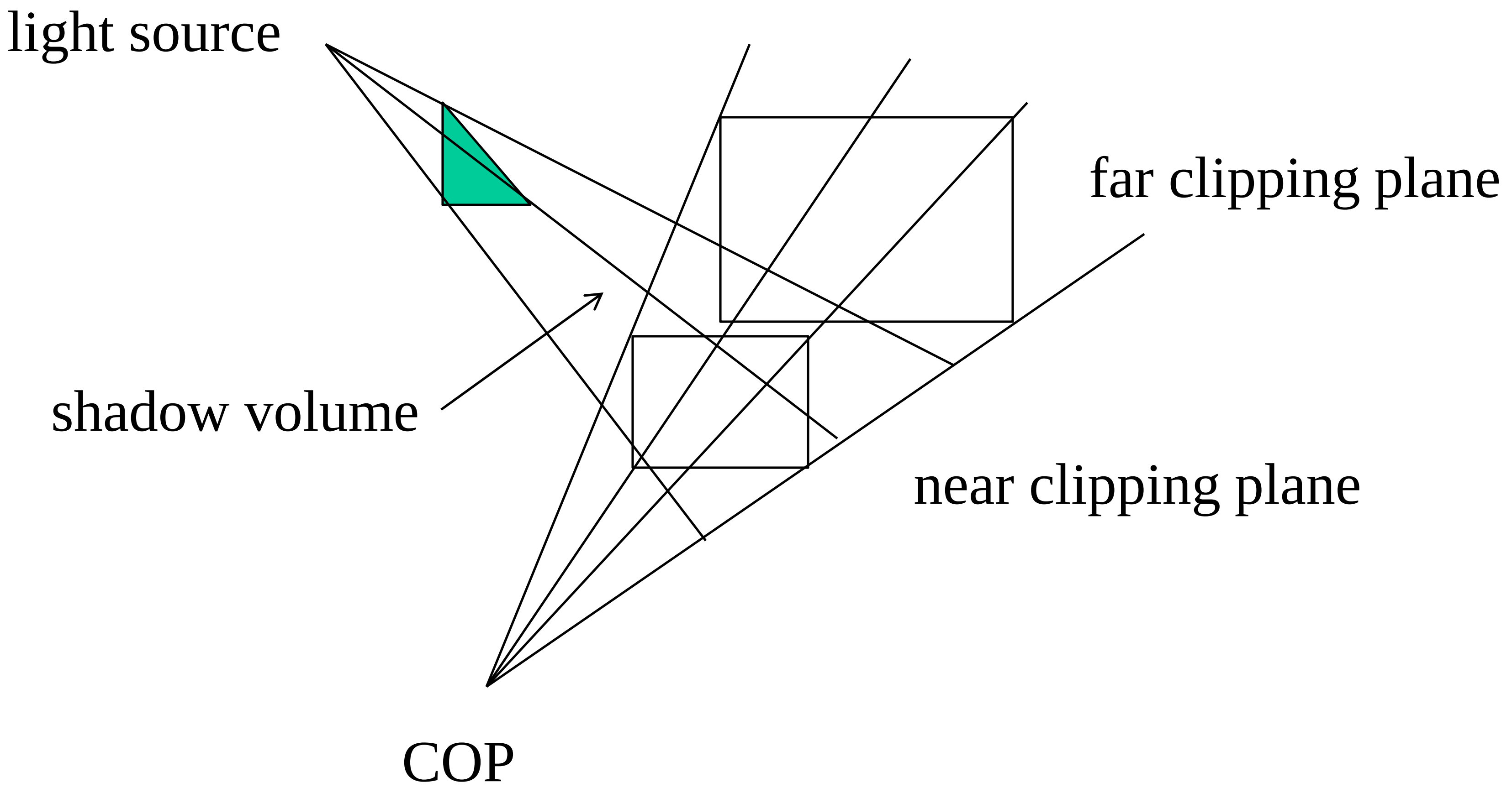
Final Rendering

- During the final rendering we compare the distance from the fragment to the light source with the distance in the shadow map
- If the depth in the shadow map is less than the distance from the fragment to the source the fragment is in shadow (from this source)
- Otherwise we use the rendered color

Implementation

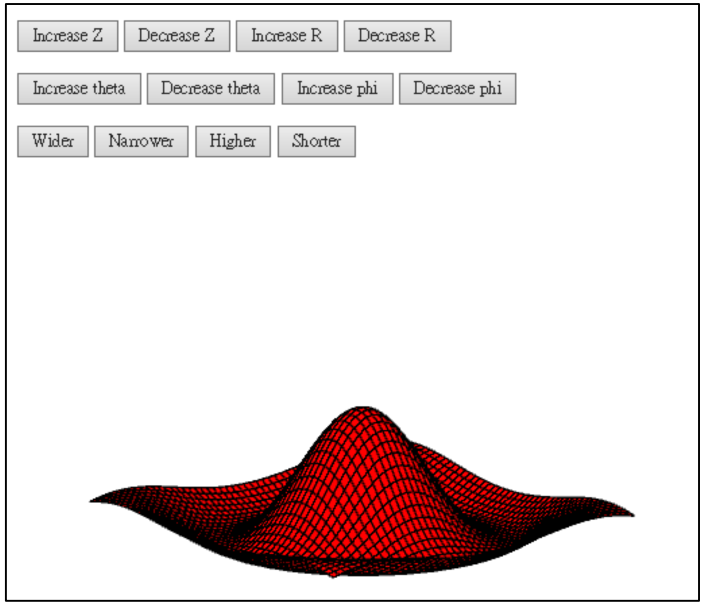
- Requires multiple renderings
- We will look at render-to-texture later
  - gives us a method to save the results of a rendering as a texture
  - almost all work done in the shaders

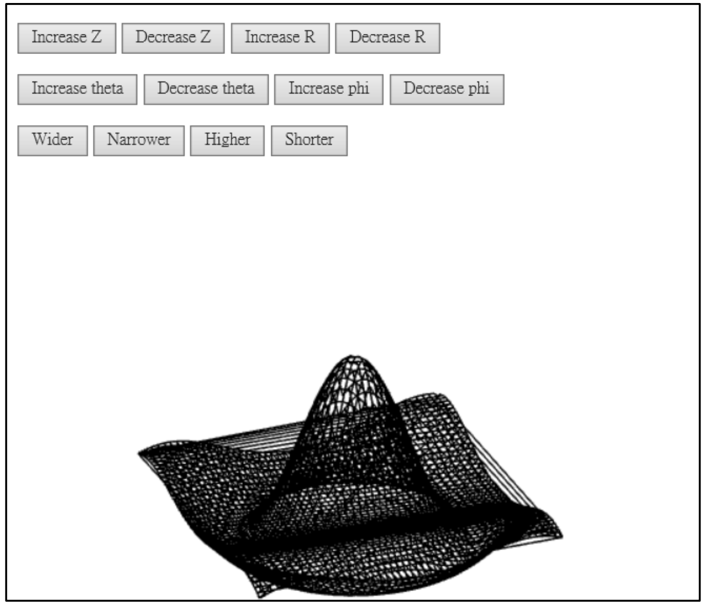
Shadow Volumes



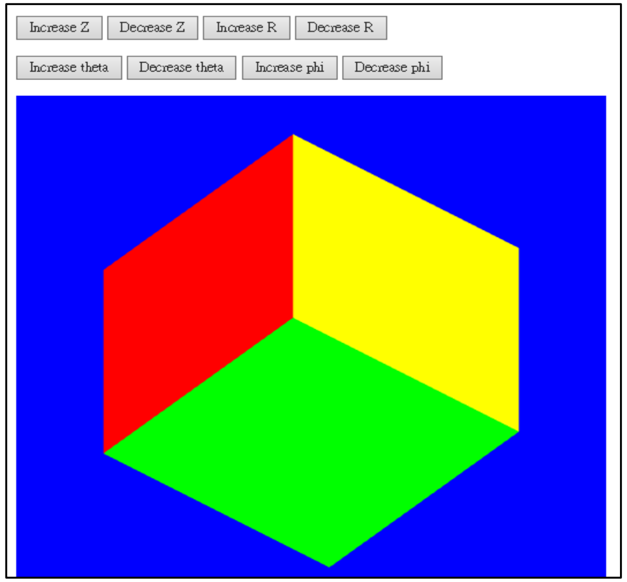
Sample Programs

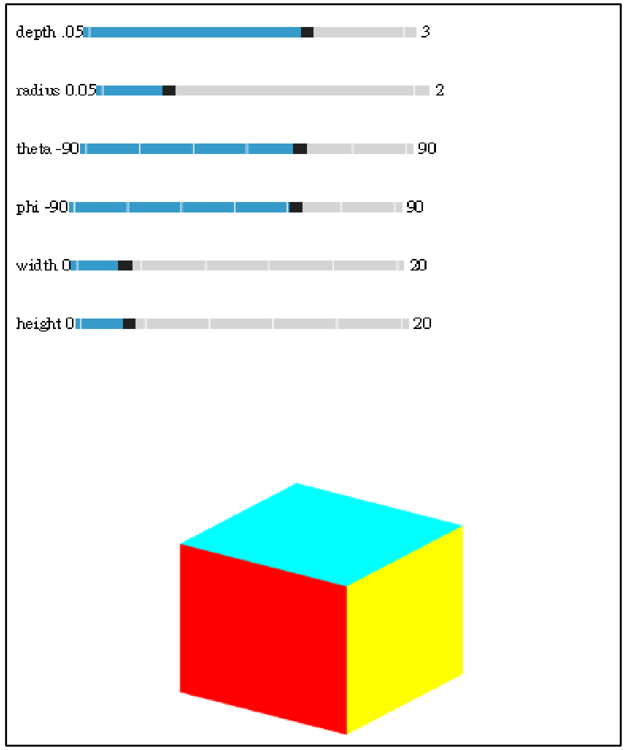
sombrero or Mexican hat function sin(πr)/(πr)

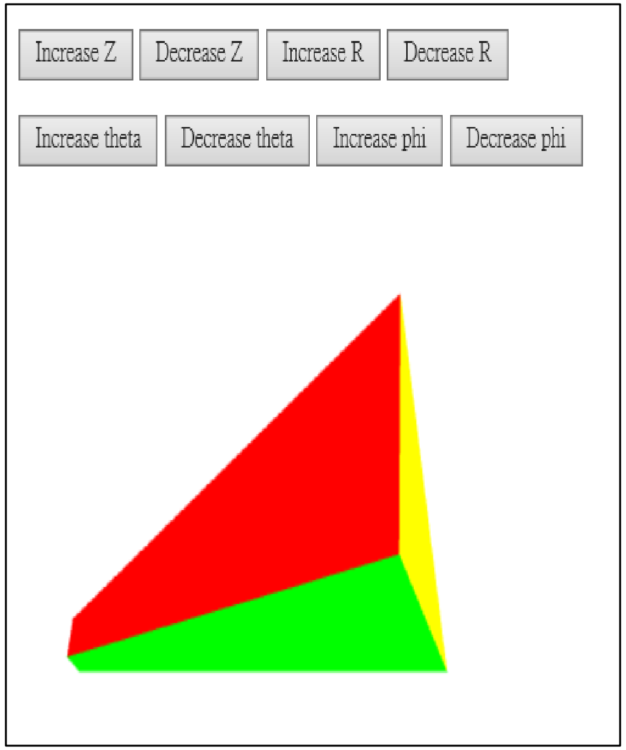


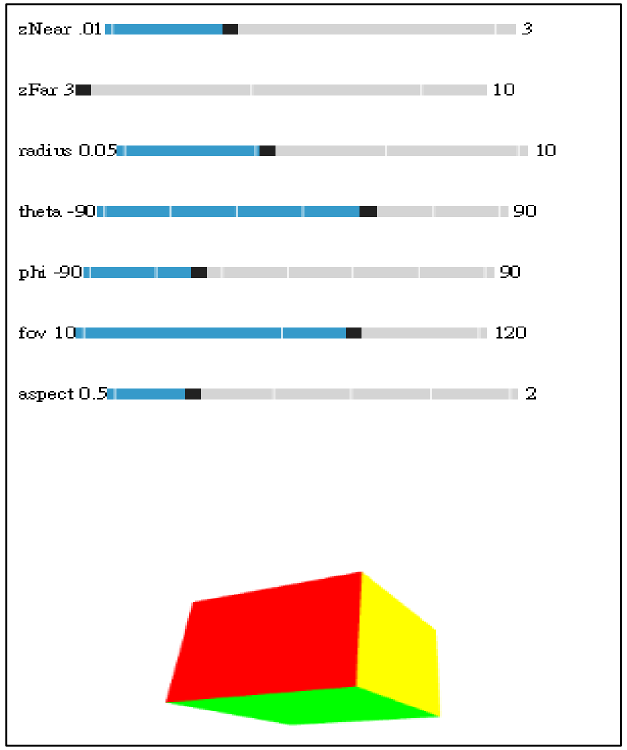


Sample Programs





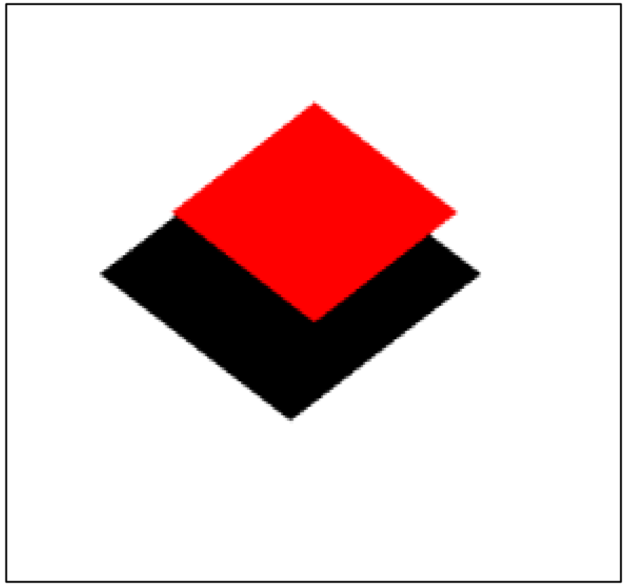




Orthographic view

Perspective view

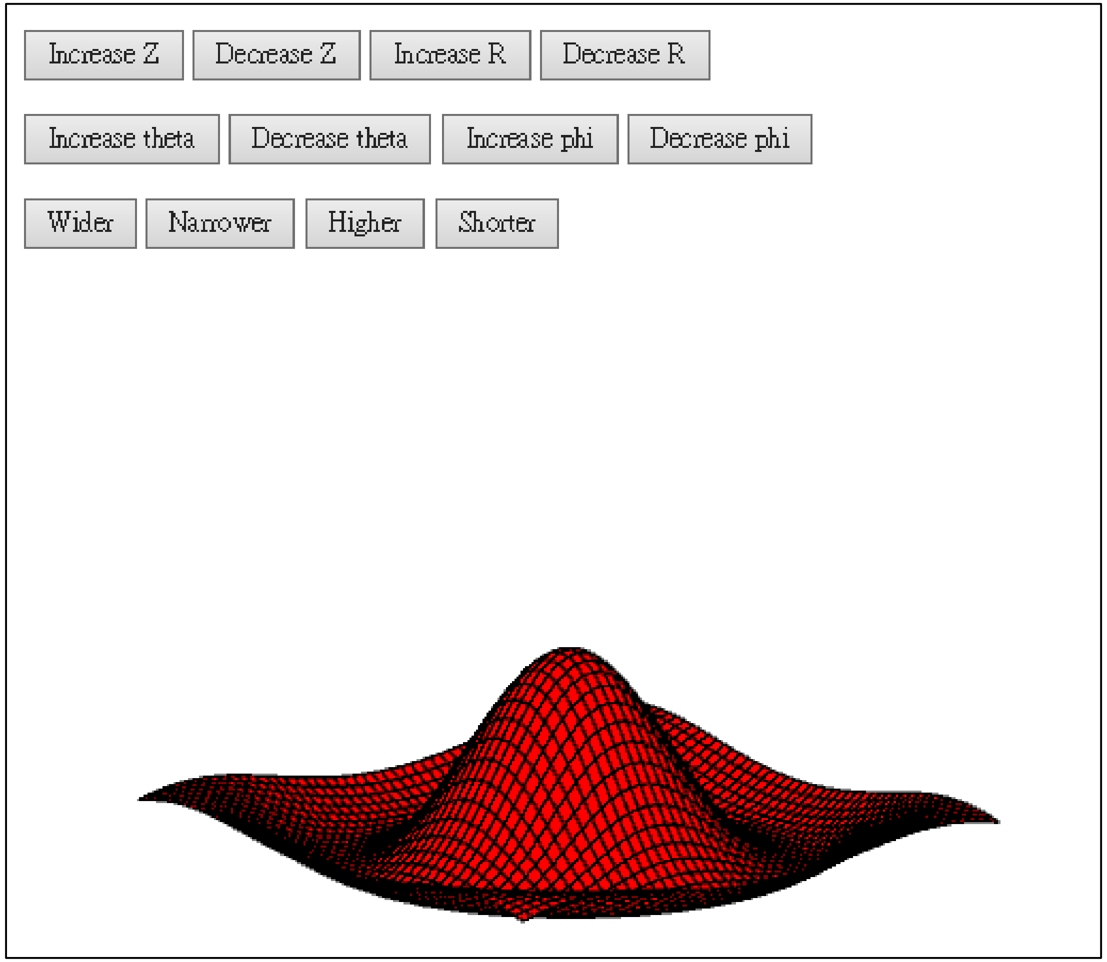
Sample Programs



Shadow polygon projection

Sample Programs: hat.html, hat.js

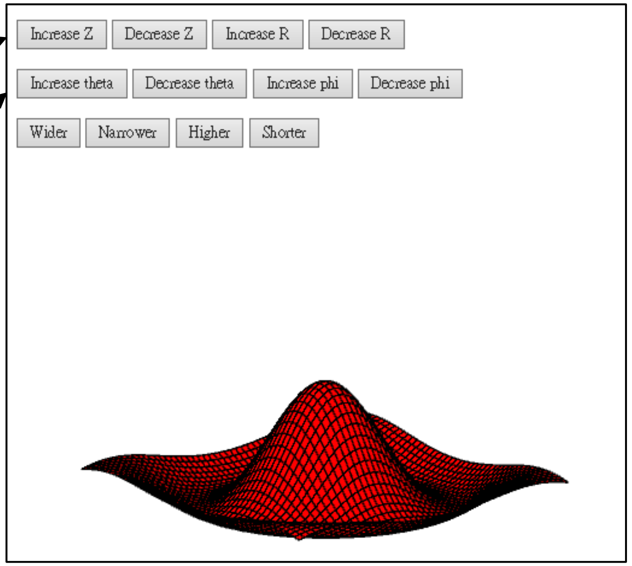
sombrero or Mexican hat function sin(πr)/(πr)



Display of sombrero function using both filled triangles and line loops

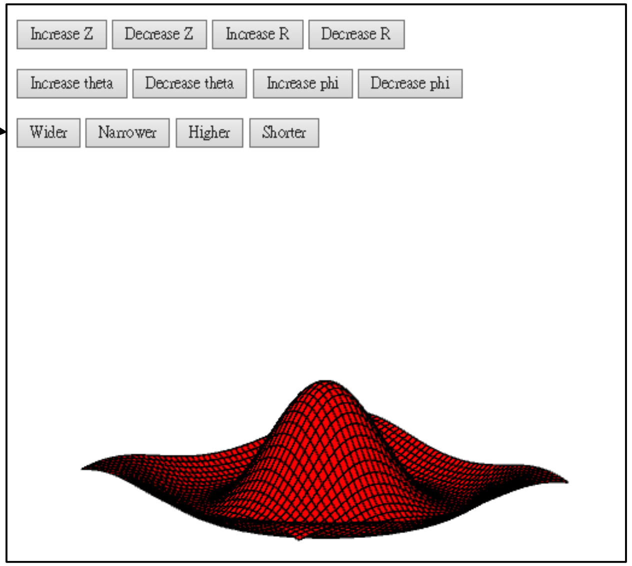
hat.html (1/4)

```
<!DOCTYPE html>
<html>

<p> </p>
<button id = "Button1">Increase Z</button>
<button id = "Button2">Decrease Z</button>
<button id = "Button3">Increase R</button>
<button id = "Button4">Decrease R</button>

<p> </p>
<button id = "Button5">Increase theta</button>
<button id = "Button6">Decrease theta</button>
<button id = "Button7">Increase phi</button>
<button id = "Button8">Decrease phi</button>
<p> </p>
```



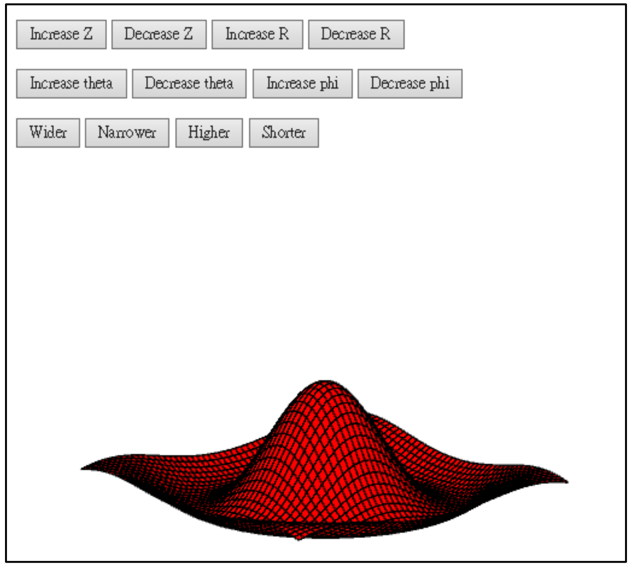
hat.html (2/4)

```
<button id = "Button9">Wider</button>
<button id = "Button10">Narrower</button>
<button id = "Button11">Higher</button>
<button id = "Button12">Shorter</button>

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

attribute  vec4 vPosition;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

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



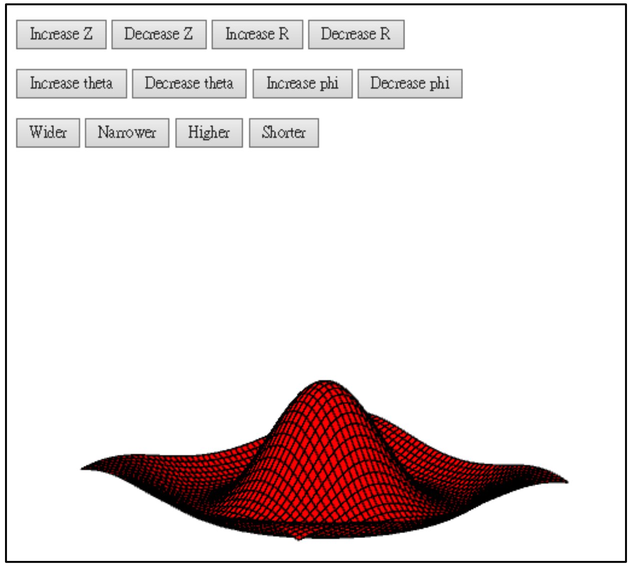
hat.html (3/4)

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

precision mediump float;

uniform 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="hat.js"></script>
```



hat.html (4/4)

```
<body>

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

</body>
</html>
```

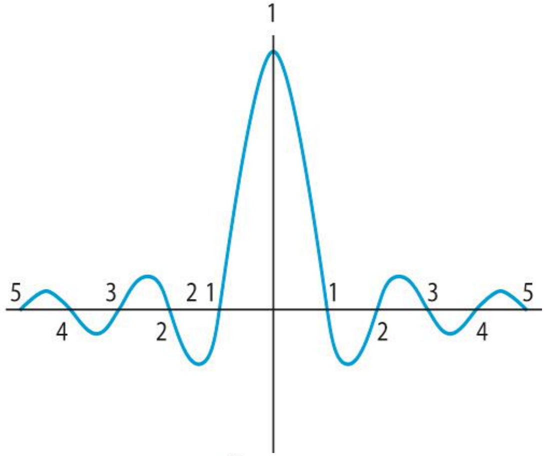


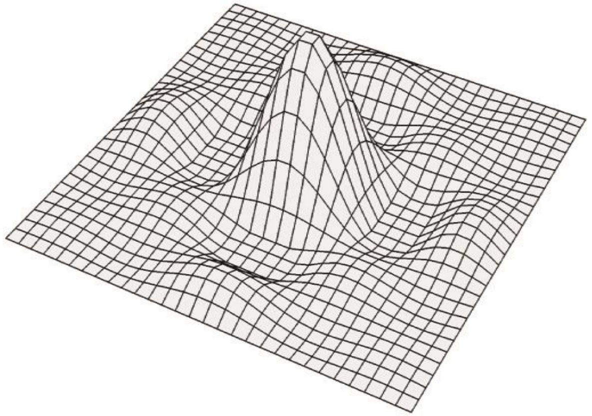
hat.js (1/11)

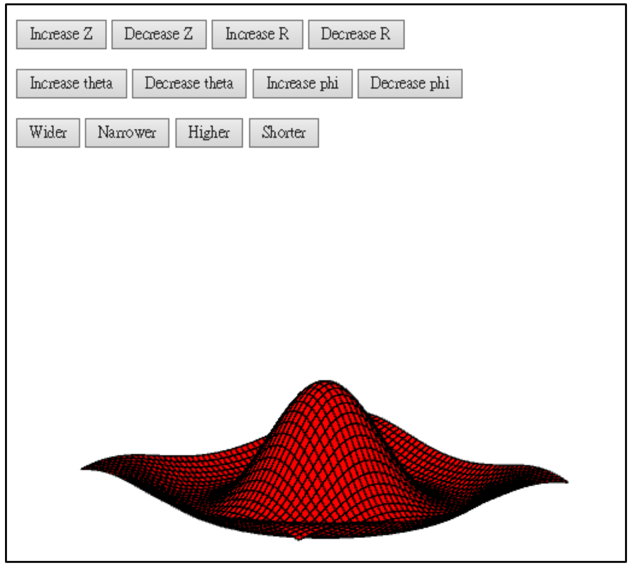
sombrero or Mexican hat function $f(r) = \sin(\pi r)/(\pi r)$

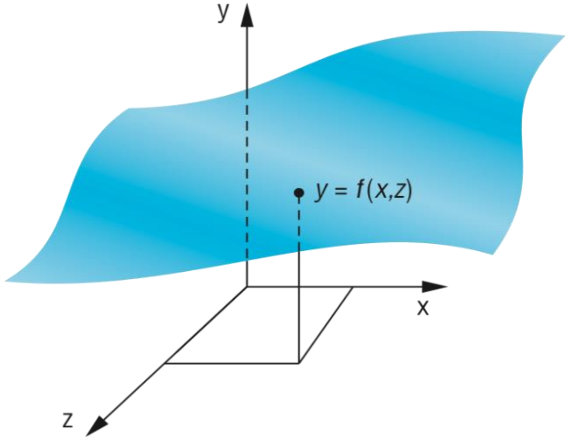
where $r = \sqrt{x^2 + z^2}$

```
var gl;

var nRows = 50;
var nColumns = 50;

// data for radial hat function: sin(Pi*r)/(Pi*r)
var data = [];
for( var i = 0; i < nRows; ++i ) {
    data.push( [] );
    var x = Math.PI*(4*i/nRows-2.0);

    for( var j = 0; j < nColumns; ++j ) {
        var y = Math.PI*(4*j/nRows-2.0);
        var r = Math.sqrt(x*x+y*y);

        // take care of 0/0 for r = 0
        data[i][j] = r ? Math.sin(r) / r : 1.0;
    }
}
```



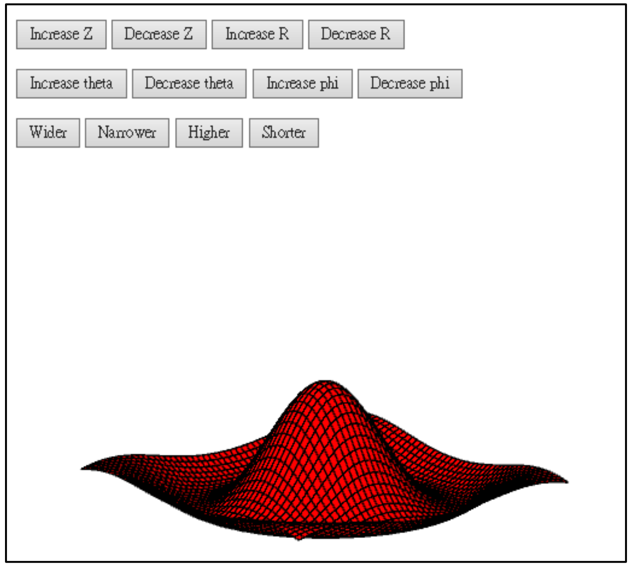






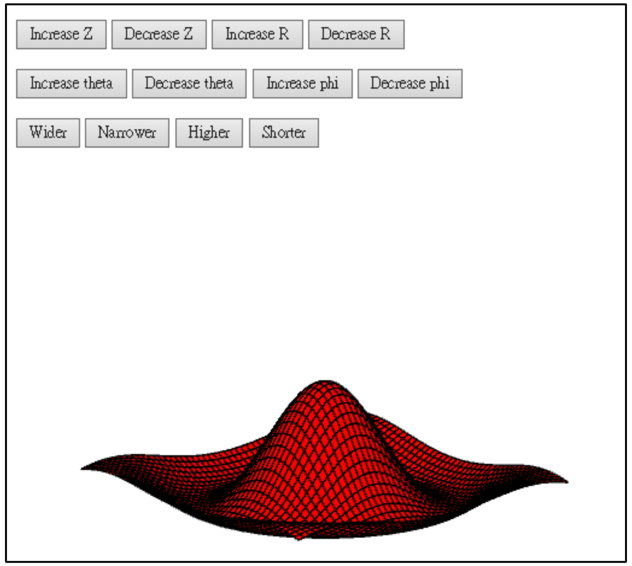
hat.js (2/11)

```
var pointsArray = [];

var fColor;

var near = -10;
var far = 10;
var radius = 6.0;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;
```



hat.js (3/11)

```
const black = vec4(0.0, 0.0, 0.0, 1.0);
const red = vec4(1.0, 0.0, 0.0, 1.0);

const at  = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);

var left = -2.0;
var right = 2.0;
var ytop = 2.0;
var bottom = -2.0;

var modeViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
```



hat.js (4/11)

```
window.onload = function init()
{
    var 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 );

    // enable depth testing and polygon offset
    // so lines will be in front of filled triangles

    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);
    gl.enable(gl.POLYGON_OFFSET_FILL);
    gl.polygonOffset(1.0, 2.0);
```

specify a function that compares incoming pixel depth to the current depth buffer value.

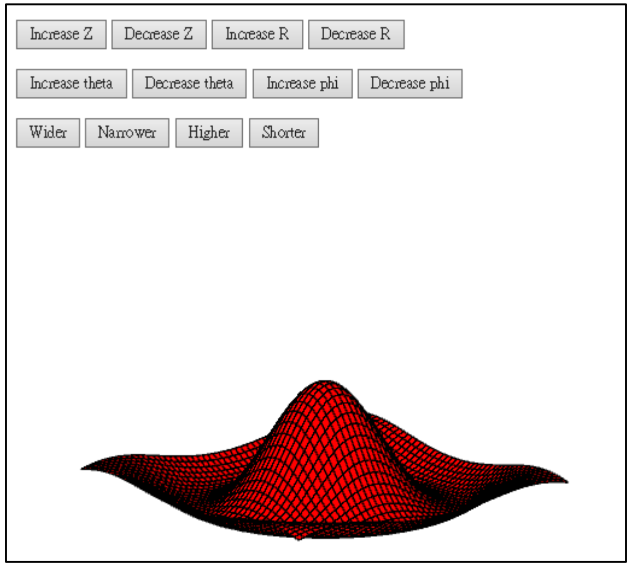
pass if the incoming value is less than or equal to the depth buffer value

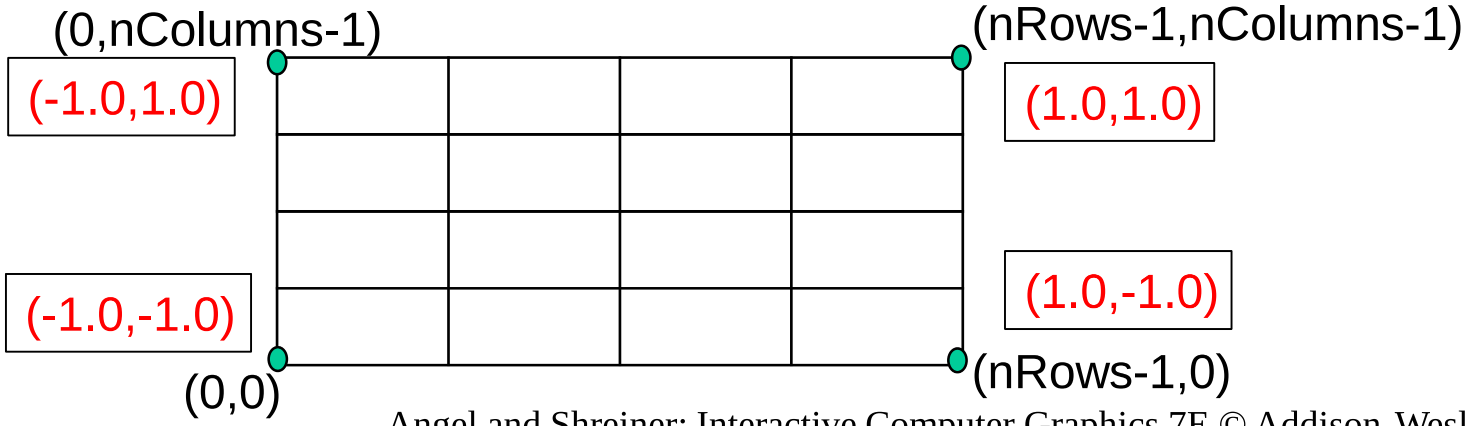
activate adding an offset to depth values of polygon's fragments

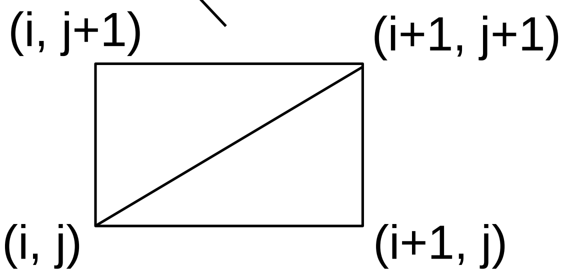
specify the scale factor (1.0) and unit (2.0) to calculate depth values.

hat.js (5/11)

```
// vertex array of nRows*nColumns quadrilaterals
// (two triangles/quad) from data

 for(var i=0; i<nRows-1; i++) {
    for(var j=0; j<nColumns-1;j++) {
      pointsArray.push( vec4(2*i/nRows-1,       data[i][j],       2*j/nColumns-1,      1.0));
      pointsArray.push( vec4(2*(i+1)/nRows-1, data[i+1][j],     2*j/nColumns-1,      1.0));
      pointsArray.push( vec4(2*(i+1)/nRows-1, data[i+1][j+1], 2*(j+1)/nColumns-1, 1.0));
      pointsArray.push( vec4(2*i/nRows-1,       data[i][j+1],     2*(j+1)/nColumns-1, 1.0));
    }
 }
```





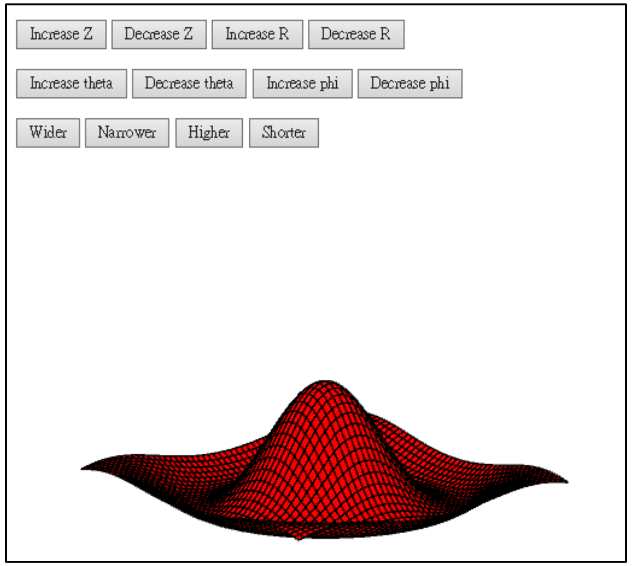


gl.TRIANGLE_FAN

gl.LINE_LOOP

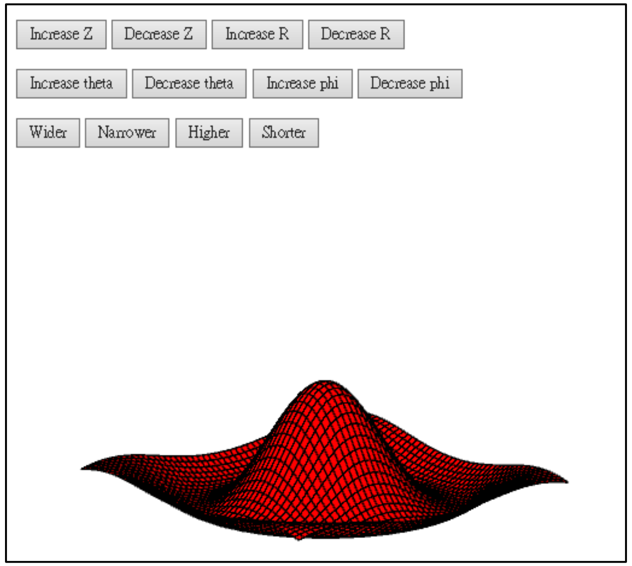
hat.js (6/11)

```
//  Load shaders and initialize attribute buffers

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

var vBufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, vBufferId );
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 );

fColor = gl.getUniformLocation(program, "fColor");

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



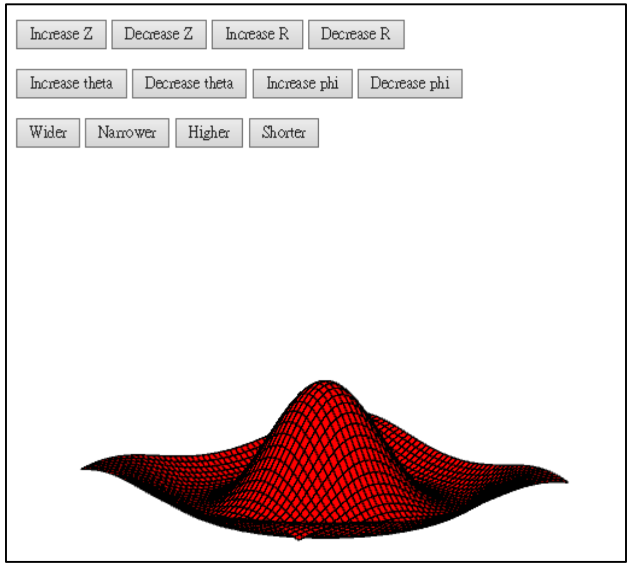
hat.js (7/11)

```
// buttons for moving viewer and changing size

    document.getElementById("Button1").onclick = function() {near  *= 1.1; far *= 1.1;};
    document.getElementById("Button2").onclick = function() {near  *= 0.9; far *= 0.9;};
    document.getElementById("Button3").onclick = function() {radius *= 2.0;};
    document.getElementById("Button4").onclick = function() {radius *= 0.5;};
    document.getElementById("Button5").onclick = function() {theta += dr;};
    document.getElementById("Button6").onclick = function() {theta -= dr;};
    document.getElementById("Button7").onclick = function() {phi += dr;};
    document.getElementById("Button8").onclick = function() {phi -= dr;};
    document.getElementById("Button9").onclick = function() {left  *= 0.9; right *= 0.9;};
    document.getElementById("Button10").onclick = function() {left *= 1.1; right *= 1.1;};
    document.getElementById("Button11").onclick = function() {ytop  *= 0.9; bottom *= 0.9;};
    document.getElementById("Button12").onclick = function() {ytop *= 1.1; bottom *= 1.1;};

    render();

}   // end of window.onload
```



hat.js (8/11)

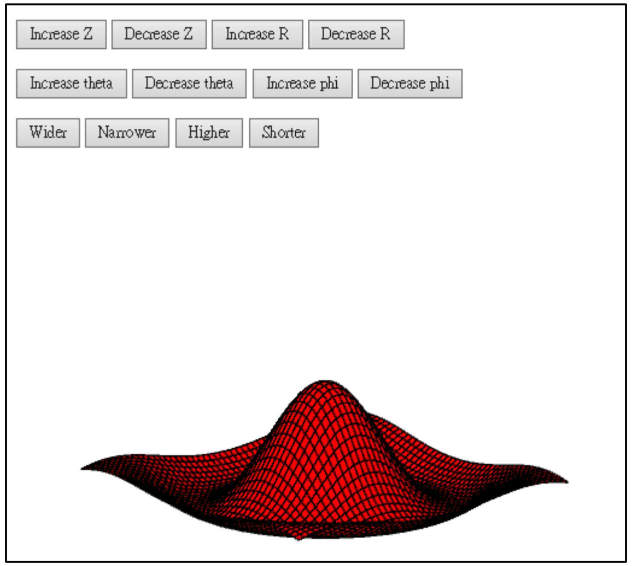
```
function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    var eye = vec3( radius*Math.sin(theta)*Math.cos(phi),
                    radius*Math.sin(theta)*Math.sin(phi),
                    radius*Math.cos(theta));

    modelViewMatrix = lookAt( eye, at, up );
    projectionMatrix = ortho( left, right, bottom, ytop, near, far );

    gl.uniformMatrix4fv( modelViewMatrixLoc, false, flatten(modelViewMatrix) );
    gl.uniformMatrix4fv( projectionMatrixLoc,   false, flatten(projectionMatrix) );
```



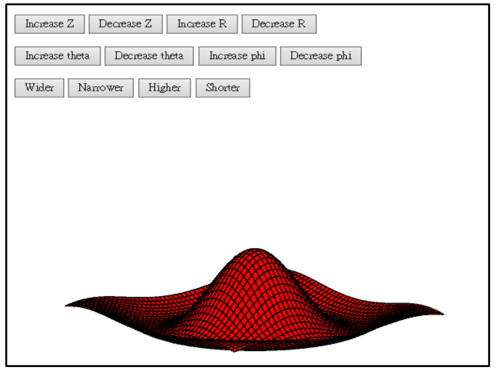
$$\mathrm{eye}(\mathrm{r}, \theta, \emptyset) \equiv \mathrm{eye}(\mathrm{x}, \mathrm{y}, \mathrm{z}) \begin{cases} \mathrm{x} = \mathrm{r} \sin\theta \cos\emptyset \\ \mathrm{y} = \mathrm{r} \sin\theta \sin\emptyset \\ \mathrm{z} = \mathrm{r} \cos\theta \end{cases}$$

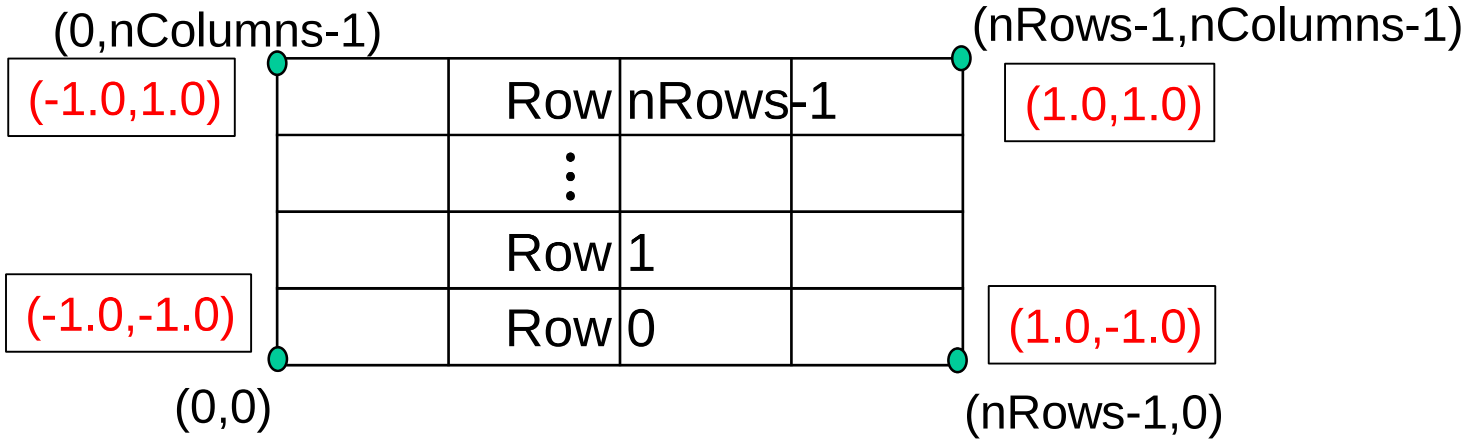
hat.js (9/11)

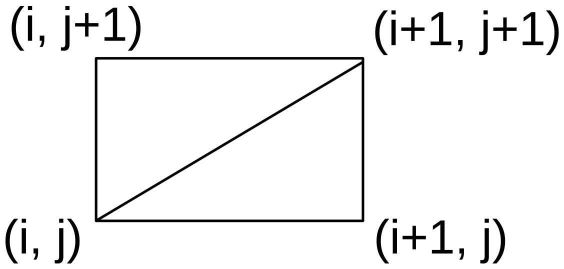
```
// draw each quad as two filled red triangles
// and then as two black line loops

for(var i=0; i<pointsArray.length; i+=4) {
    gl.uniform4fv(fColor, flatten(red));
    gl.drawArrays( gl.TRIANGLE_FAN, i, 4 );
    gl.uniform4fv(fColor, flatten(black));
    gl.drawArrays( gl.LINE_LOOP, i, 4 );
 }

requestAnimFrame(render);
}   // end of render()
```



A note on hat.js: pointsArray Data Structure (10/11)



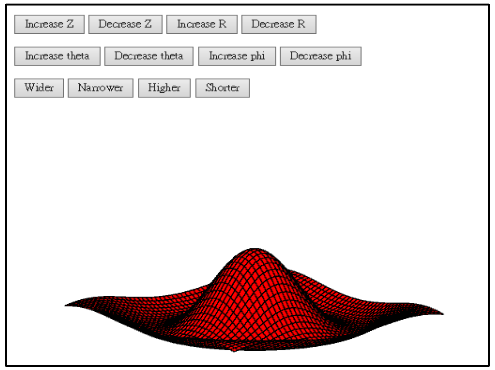




gl.TRIANGLE_FAN

gl.LINE_LOOP

A note on hat.js: pointsArray Data Structure (11/11)



pointsArray

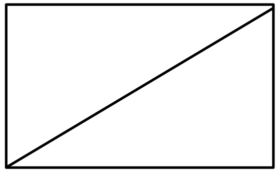
	0	4	8	4(nClumns-1)
Row 0	4 points	4 points	…	4 points

	4(nColumns)	4(nColumns+1)	4(nColumns+2)	4(2*nClumns-1)
Row 1	4 points	4 points	…	4 points

⋮ ⋮

	4((nRows-1)*nColumns)	4((nRows-1)*nColumns+1)	4((nRows-1)*nColumns+2)	4(nRows**nClumns-1)
Row nRows-1	4 points	4 points	…	4 points

(i, j+1) (i+1, j+1)



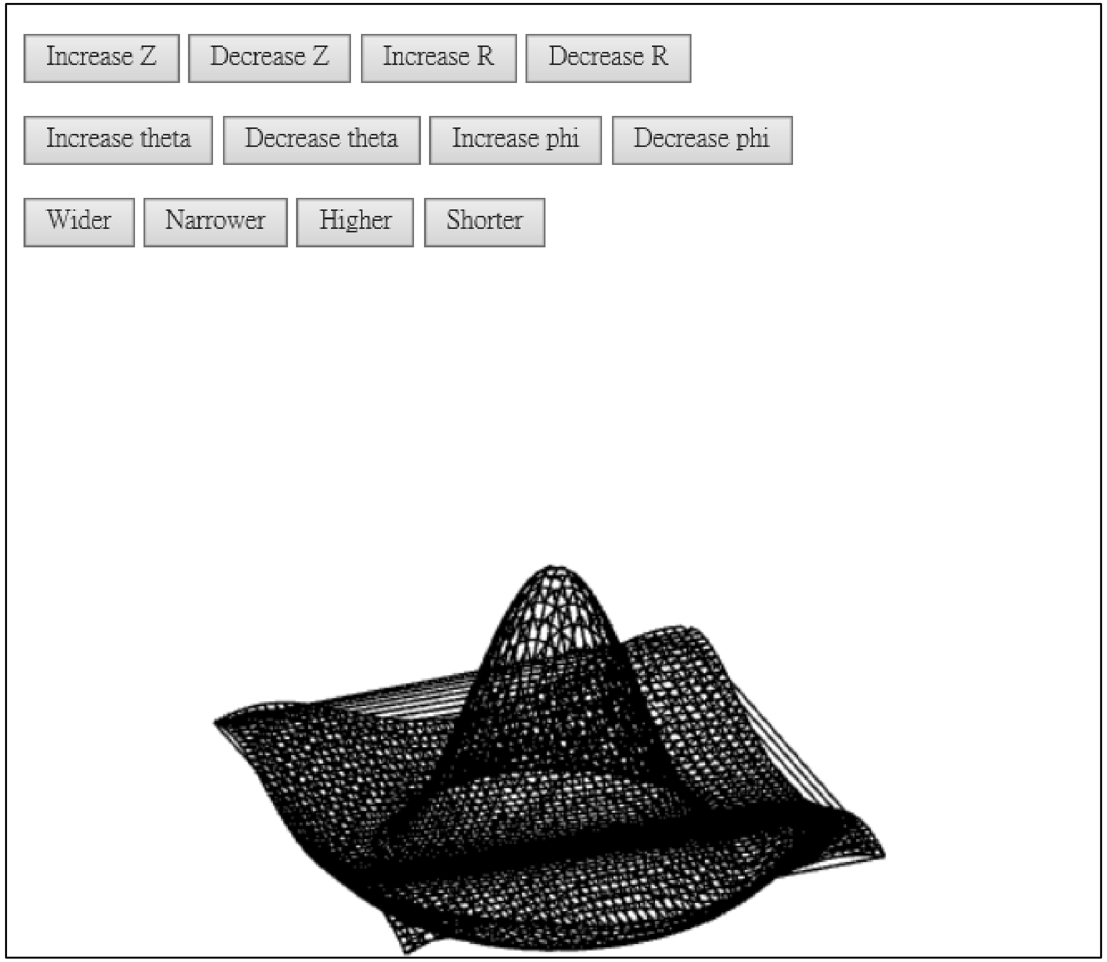
(i, j) (i+1, j)

gl.TRIANGLE_FAN
gl.LINE_LOOP

4 points:
(i,j), (i+1,j), (i+1,j+1), (i,j+1)

Sample Programs: hata.html, hata.js

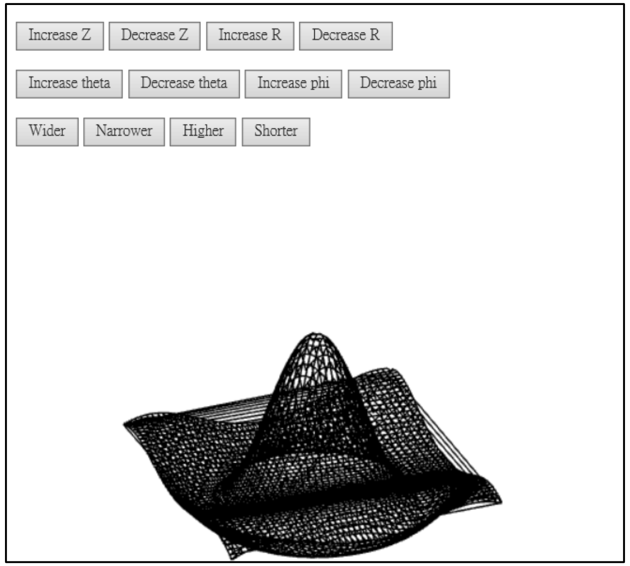
sombrero or Mexican hat function sin(πr)/(πr)



Display of sombrero function
using line strips in two directions

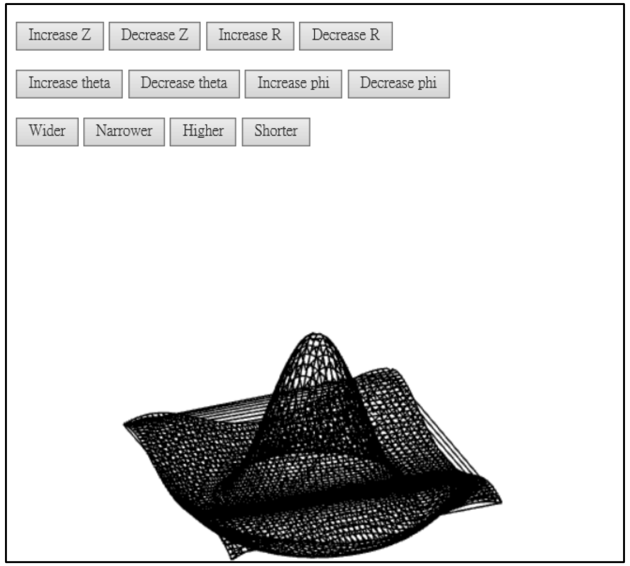
hata.html (1/4)

```
<!DOCTYPE html>
<html>

<p> </p>
<button id = "Button1">Increase Z</button>
<button id = "Button2">Decrease Z</button>
<button id = "Button3">Increase R</button>
<button id = "Button4">Decrease R</button>

<p> </p>
<button id = "Button5">Increase theta</button>
<button id = "Button6">Decrease theta</button>
<button id = "Button7">Increase phi</button>
<button id = "Button8">Decrease phi</button>
```



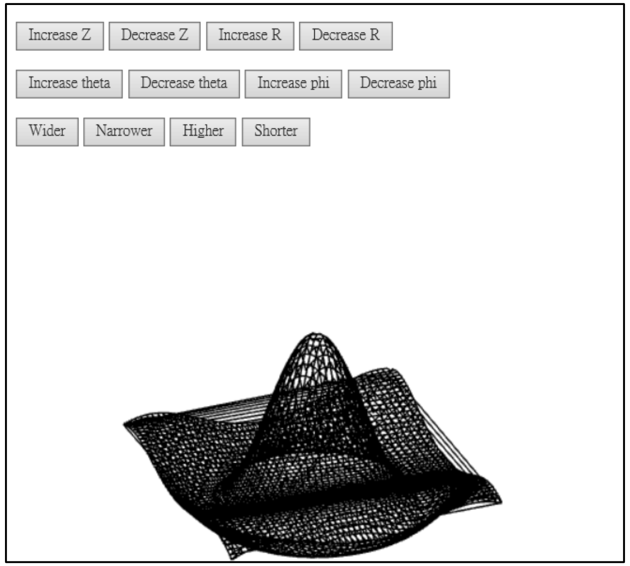
hata.html (2/4)

```
<button id = "Button9">Wider</button>
<button id = "Button10">Narrower</button>
<button id = "Button11">Higher</button>
<button id = "Button12">Shorter</button>

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

attribute  vec4 vPosition;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

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



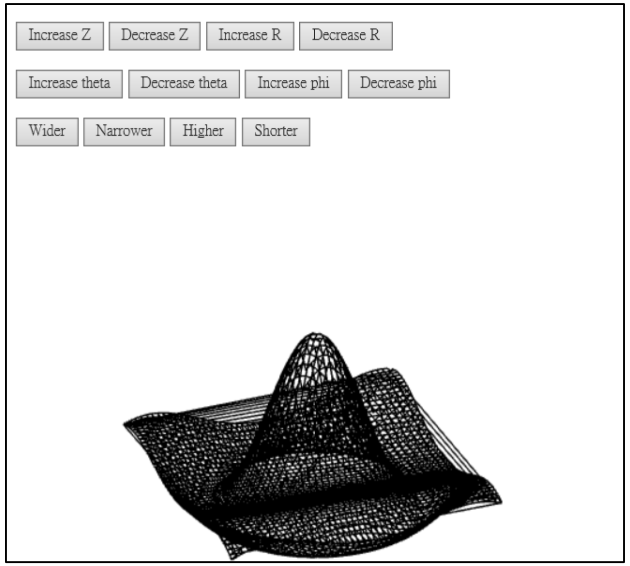
hata.html (3/4)

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

precision mediump float;

//uniform vec4 fColor;

void
main()
{
   gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
}
</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="hata.js"></script>
```



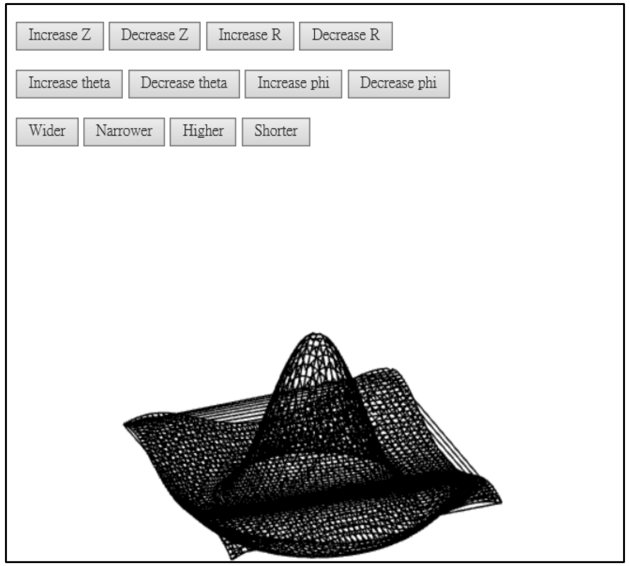
hata.html (4/4)

```
<body>

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

</body>
</html>
```



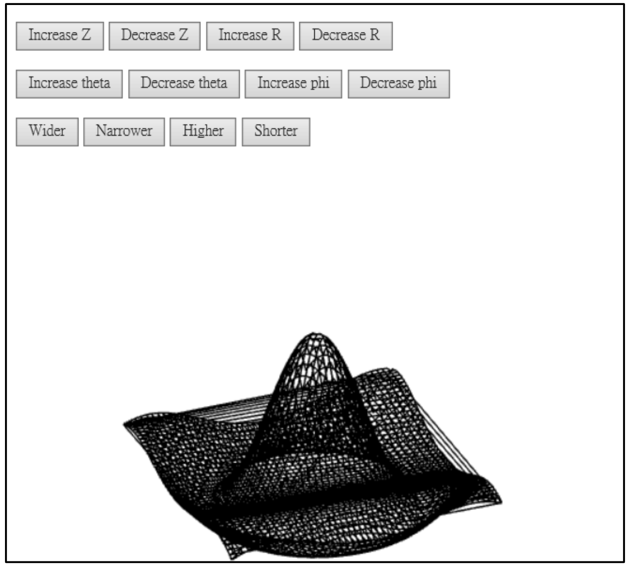
hata.js (1/9)

```
// data for radial hat function: sin(Pi*r)/(Pi*r)

var data = new Array(nRows);
for(var i =0; i<nRows; i++) data[i]=new Array(nColumns);

for(var i=0; i<nRows; i++) {
    var x = Math.PI*(4*i/nRows-2.0);
    for(var j=0; j<nColumns; j++) {
        var y = Math.PI*(4*j/nRows-2.0);
        var r = Math.sqrt(x*x+y*y)

        // take care of 0/0 for r = 0

        if(r) data[i][j] = Math.sin(r)/r;
        else data[i][j] = 1;
    }
}
```



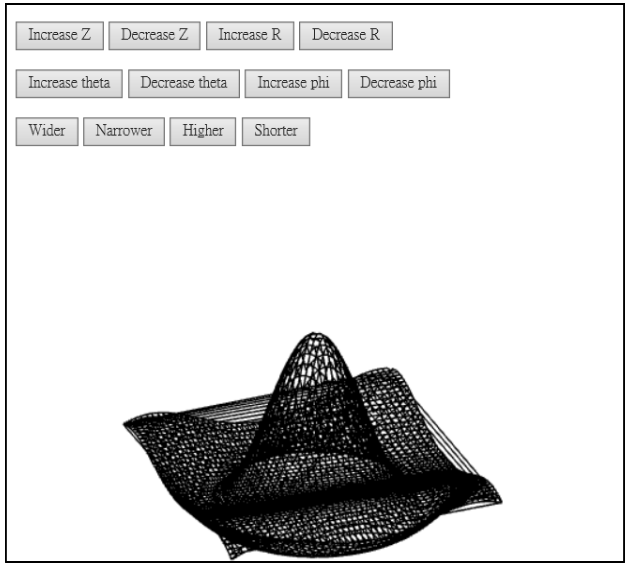
hata.js (2/9)

```
var pointsArray = [];

var canvas;
var gl;

var near = -10;
var far = 10;
var radius = 1.0;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;

var left = -2.0;
var right = 2.0;
var ytop = 2.0;
var bottom = -2.0;
```



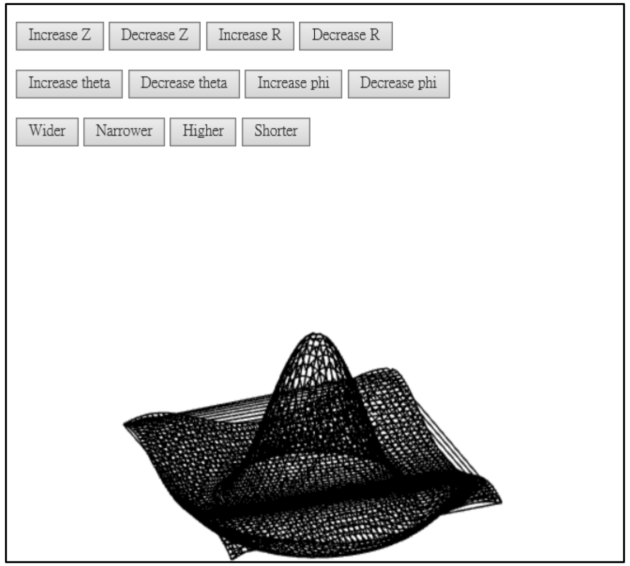
hata.js (3/9)

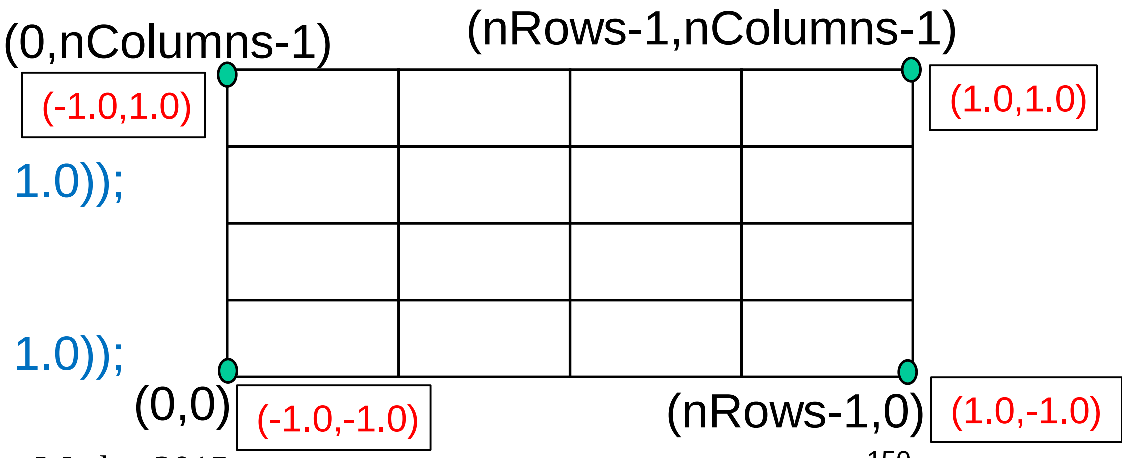
```
var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;

var eye;

const at  = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);
```



hata.js (4/9)

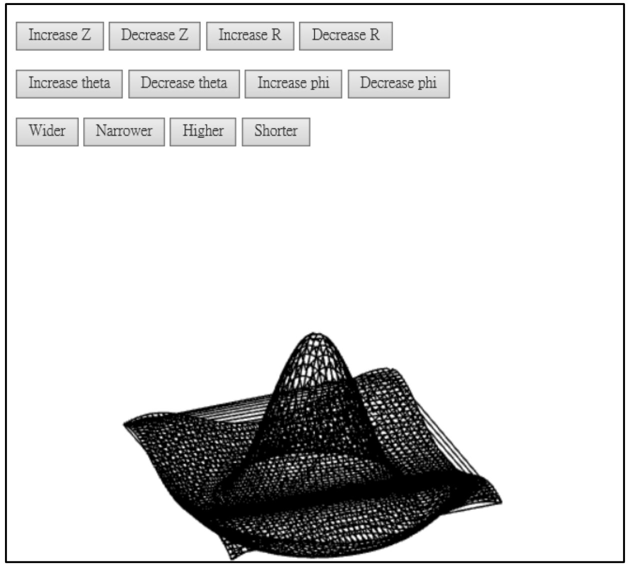
```
window.onload = function init() {
   canvas = document.getElementById( "gl-canvas" );

   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 );

  // vertex array of data for nRows and nColumns of line strips

   for(var i=0; i<nRows-1; i++) for(var j=0; j<nColumns-1;j++) {
      pointsArray.push(vec4(2*i/nRows-1, data[i][j], 2*j/nColumns-1, 1.0));
   }
   for(var j=0; j<nColumns-1; j++) for(var i=0; i<nRows-1;i++) {
      pointsArray.push(vec4(2*i/nRows-1, data[i][j], 2*j/nColumns-1, 1.0));
   }
```





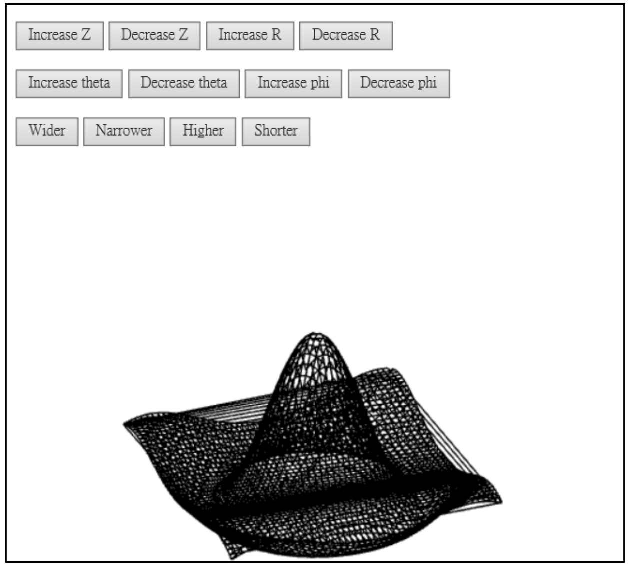
hata.js (5/9)

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

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 );

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



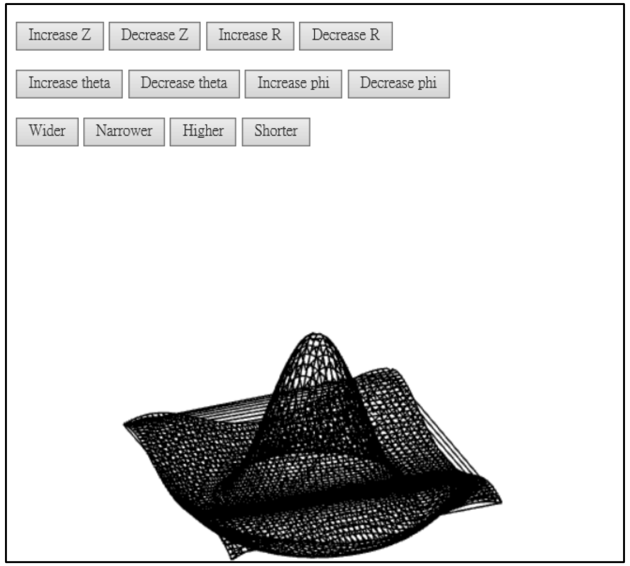
hata.js (6/9)

```
// buttons for moving viewer and changing size

    document.getElementById("Button1").onclick = function() {near  *= 1.1; far *= 1.1;};
    document.getElementById("Button2").onclick = function() {near  *= 0.9; far *= 0.9;};
    document.getElementById("Button3").onclick = function() {radius *= 2.0;};
    document.getElementById("Button4").onclick = function() {radius *= 0.5;};
    document.getElementById("Button5").onclick = function() {theta += dr;};
    document.getElementById("Button6").onclick = function() {theta -= dr;};
    document.getElementById("Button7").onclick = function() {phi += dr;};
    document.getElementById("Button8").onclick = function() {phi -= dr;};
    document.getElementById("Button9").onclick = function() {left  *= 0.9; right *= 0.9;};
    document.getElementById("Button10").onclick = function() {left *= 1.1; right *= 1.1;};
    document.getElementById("Button11").onclick = function() {ytop  *= 0.9; bottom *= 0.9;};
    document.getElementById("Button12").onclick = function() {ytop *= 1.1; bottom *= 1.1;};

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

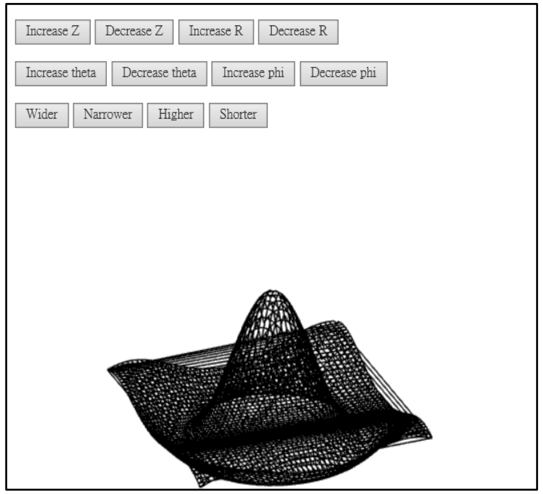


hata.js (7/9)

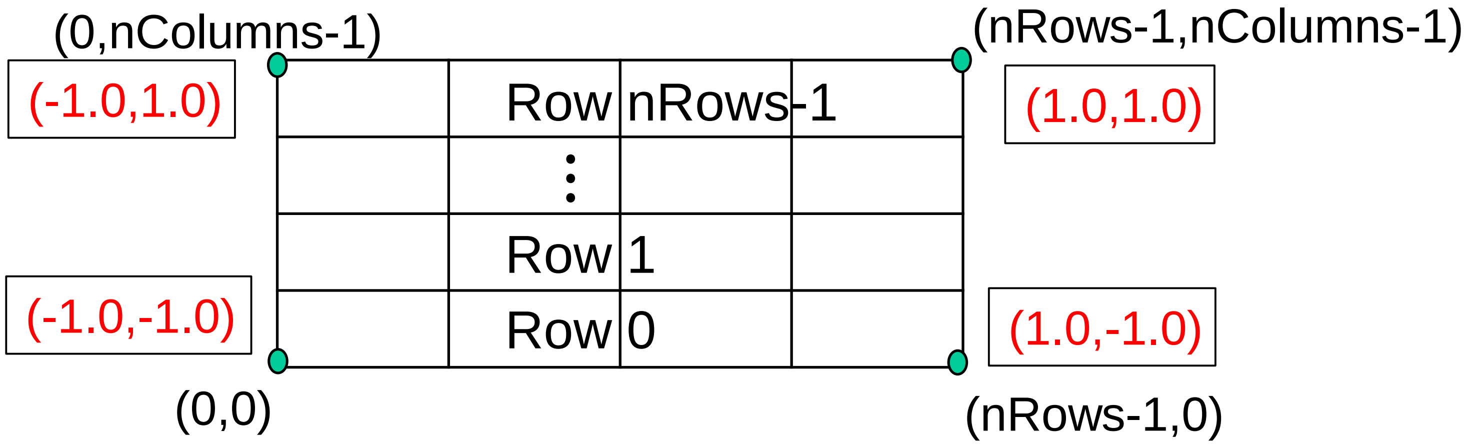
```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
                   radius*Math.sin(theta)*Math.sin(phi), radius*Math.cos(theta));

    modelViewMatrix = lookAt(eye, at , up);
    projectionMatrix   = ortho(left, right, bottom, ytop, near, far);

    gl.uniformMatrix4fv( modelViewMatrixLoc, false, flatten(modelViewMatrix) );
    gl.uniformMatrix4fv( projectionMatrixLoc,   false, flatten(projectionMatrix) );

    // render columns of data then rows
    for(var i=0; i<nRows; i++)      gl.drawArrays( gl.LINE_STRIP, i*nColumns, nColumns );
    for(var i=0; i<nColumns; i++) gl.drawArrays( gl.LINE_STRIP, i*nRows+pointsArray.length/2, nRows );

    requestAnimFrame(render);
}   // end of render()
```

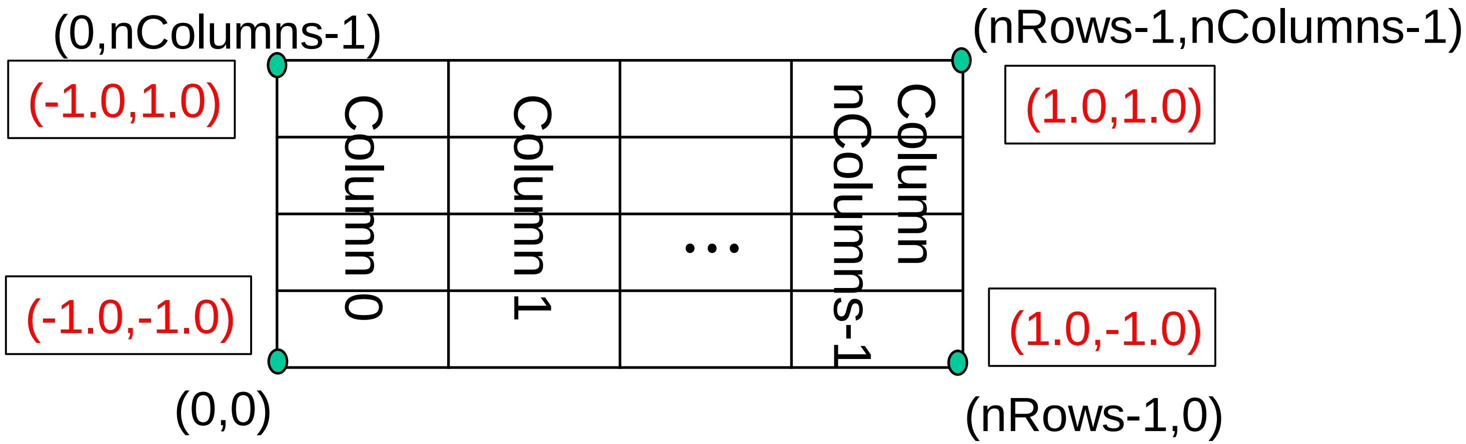


A note on hata.js: pointsArray Data Structure (8/9)

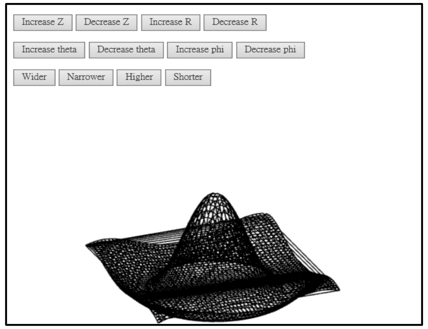


pointsArray



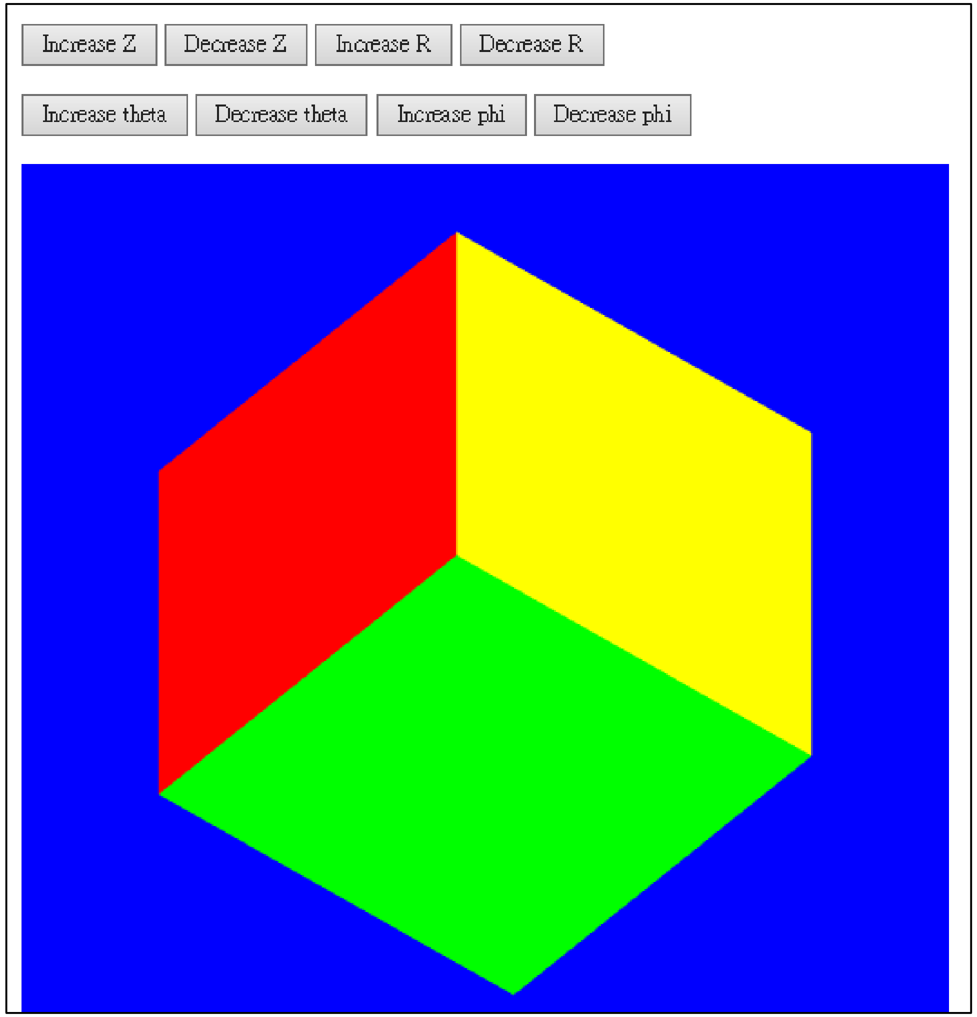


A note on hata.js: pointsArray Data Structure (9/9)



	0	1	2		nColumn-1
Row 0	(0,0)	(1,0)	(2,0)	···	(nColumns-1,0)
	nColumn	nColumn+1	nColumn+2		2*nColumn-1
Row 1	(0,1)	(1,1)	(2,1)	···	(nColumns-1,1)
⋮	(nRows-1)*nColumn		⋮		nRows*nColumn-1
Row nRows-1	(0,nRows-1)	(1,nRows-1)	(2,nRows-1)	···	(nColumns-1,1)
	nRows*nColumn				
Column 0	(0,0)	(0,1)	(0,2)	···	(0,nRows-1)
	nRows*nColumn+nRows				
Column 1	(1,0)	(1,1)	(1,2)	···	(1,nRows-1)
⋮	nRows*nColumn+(nColumns-1)*nRows		⋮		2*nRows*nColumn-1
Column nColumns-1	(nColumns-1,0)	(nColumns-1,1)	(nColumns-1,2)	···	(nColumns-1,nRows-1)

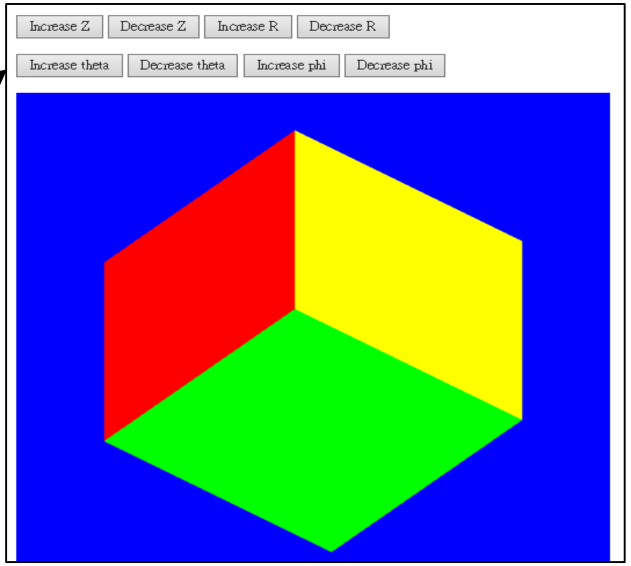
Sample Programs: ortho.html, ortho.js



Interactive orthographic viewing of cube

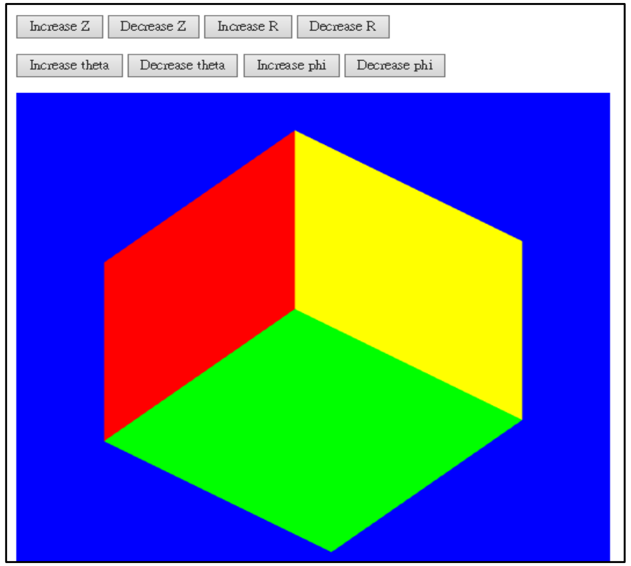
ortho.html (1/4)

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

<p> </p>
<button id = "Button1">Increase Z</button>
<button id = "Button2">Decrease Z</button>
<button id = "Button3">Increase R</button>
<button id = "Button4">Decrease R</button>

<p> </p>
<button id = "Button5">Increase theta</button>
<button id = "Button6">Decrease theta</button>
<button id = "Button7">Increase phi</button>
<button id = "Button8">Decrease phi</button>
<p> </p>
```



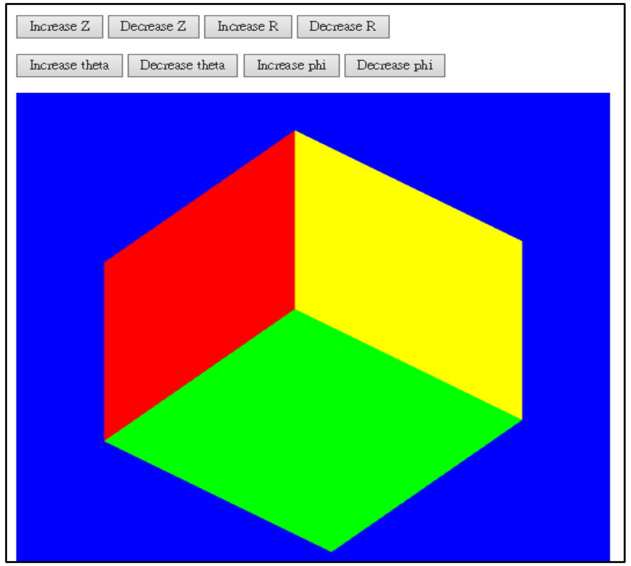
ortho.html (2/4)

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

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

uniform mat4 modelView;
uniform mat4 projection;

void main()
{
    gl_Position = projection*modelView*vPosition;
    fColor = vColor;
}
</script>
```



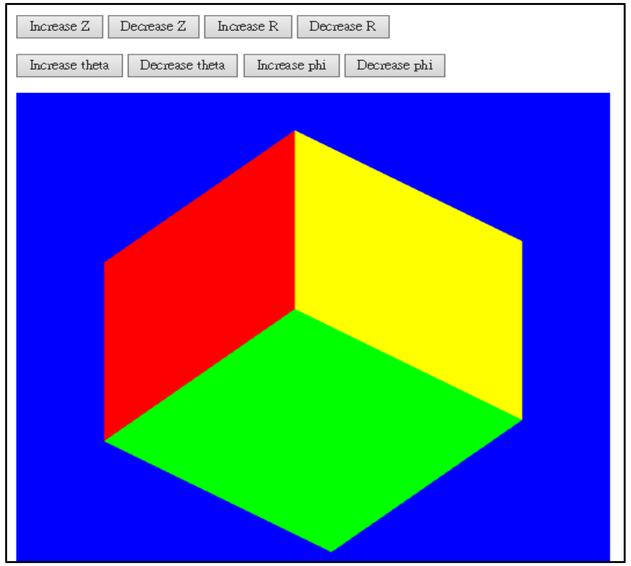
ortho.html (3/4)

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



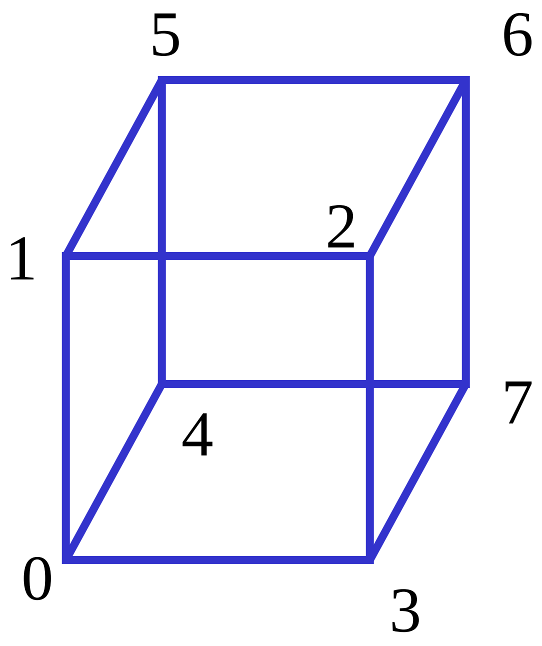
ortho.html (4/4)

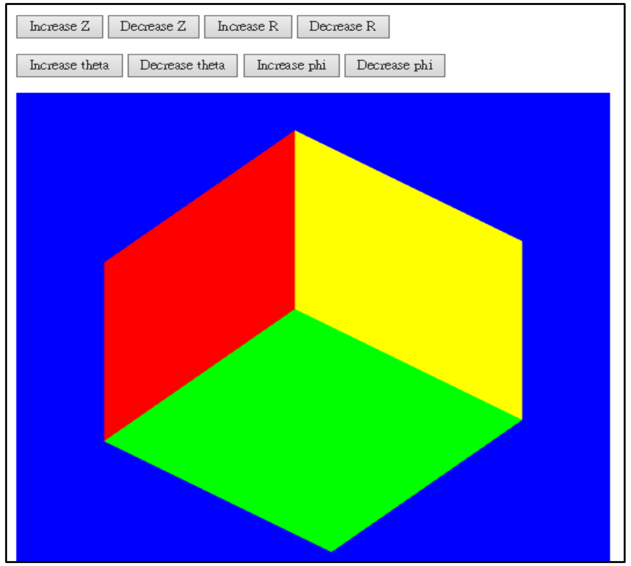
```
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>

</body>
</html>
```



ortho.js (1/10)

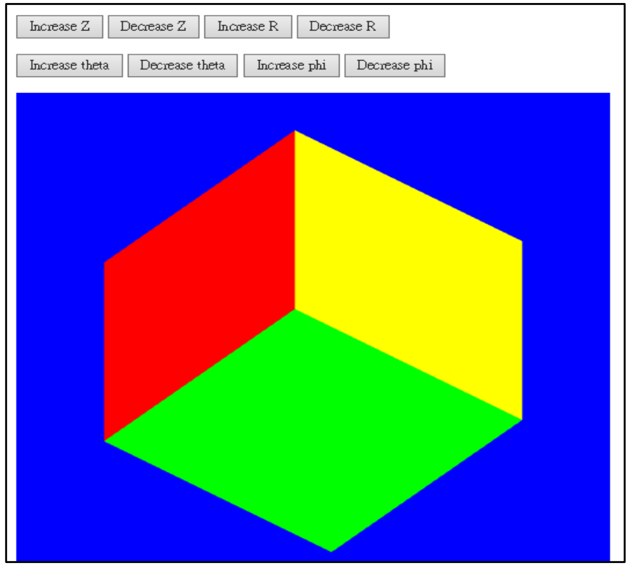
```
var canvas;
var gl;

var numVertices  = 36;

var pointsArray = [];
var colorsArray = [];

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 ),
  ];
```





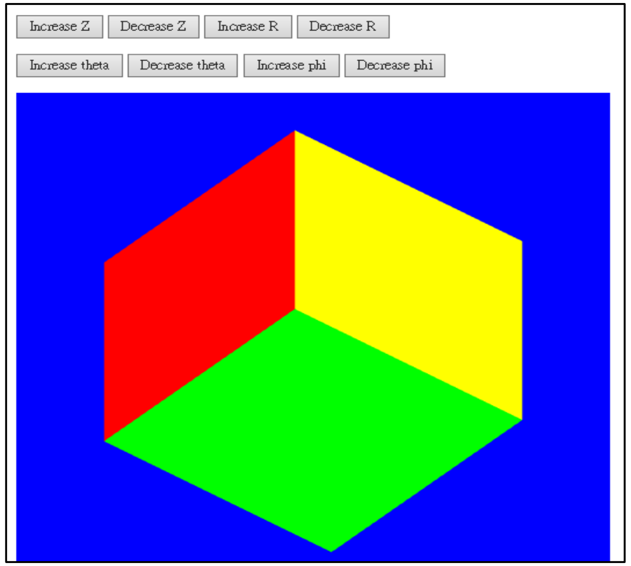
ortho.js (2/10)

```
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
    ];
```



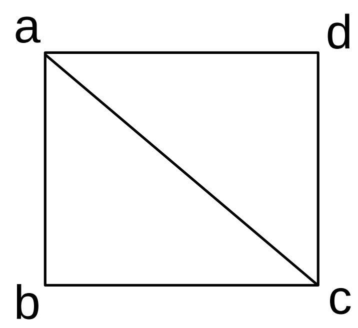
ortho.js (3/10)

```
var near = -1;
var far = 1;
var radius = 1.0;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;

var left = -1.0;
var right = 1.0;
var ytop = 1.0;
var bottom = -1.0;

var mvMatrix, pMatrix;
var modelView, projection;
var eye;

const at  = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);
```

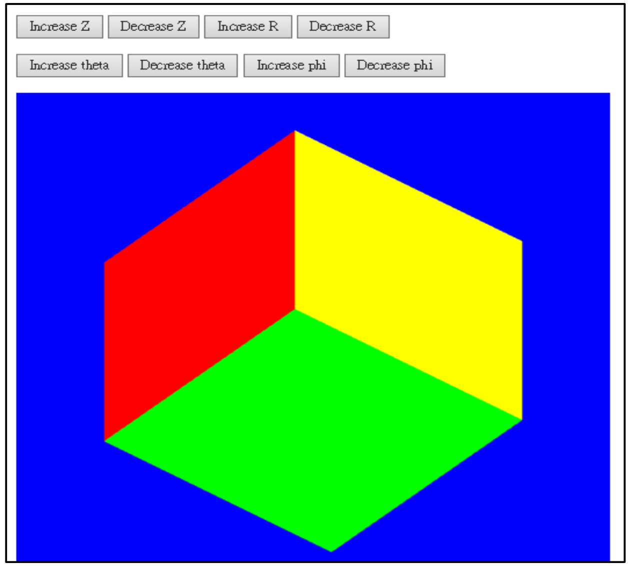


ortho.js (4/10)

```
// quad uses first index to set color for face

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

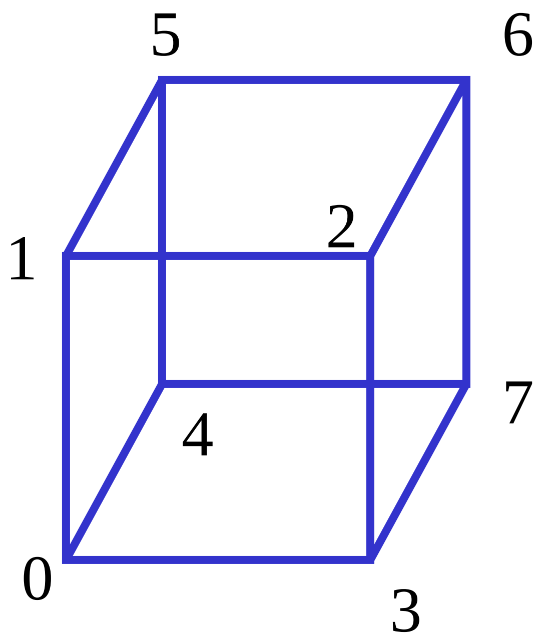


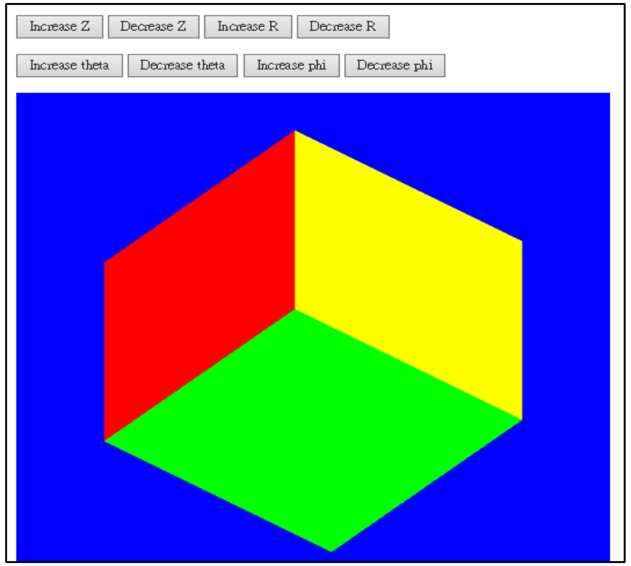
gl.TRIANGLES



ortho.js (5/10)

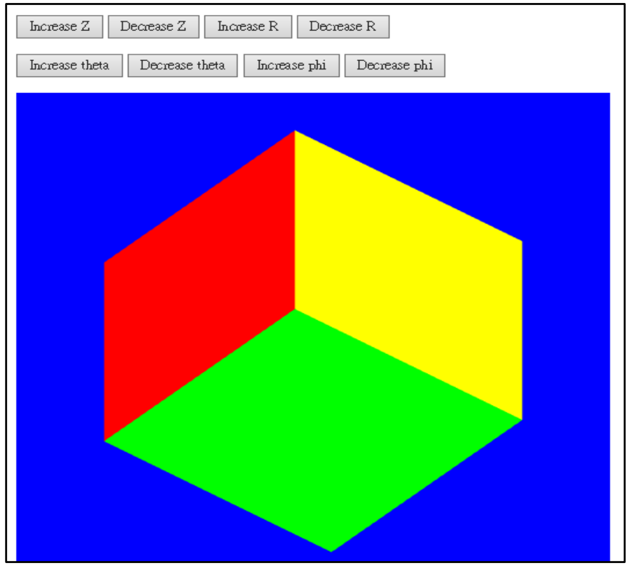
```
// Each face determines two triangles

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 );
}
```





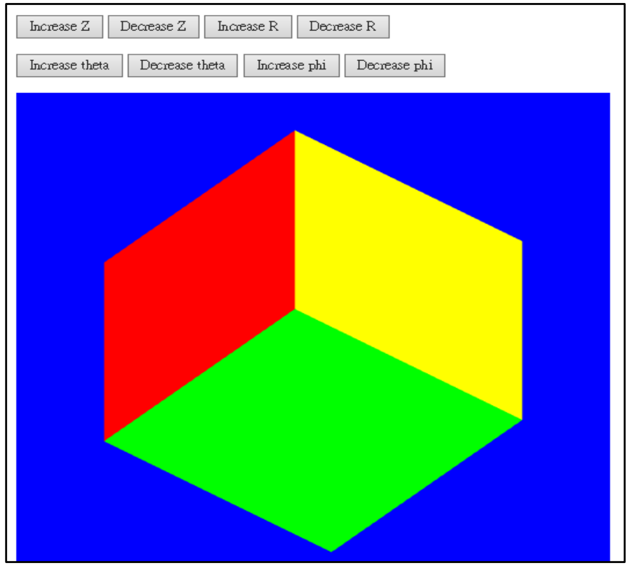
ortho.js (6/10)

```
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);
```



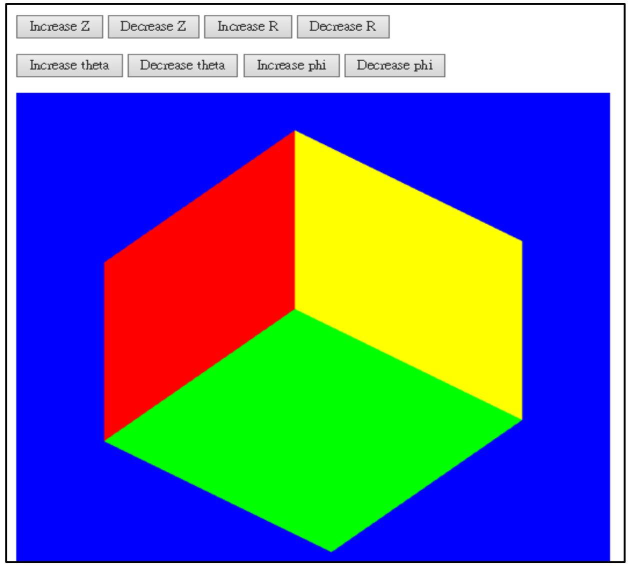
ortho.js (7/10)

```
//
//  Load shaders and initialize attribute buffers
//
var 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);
```



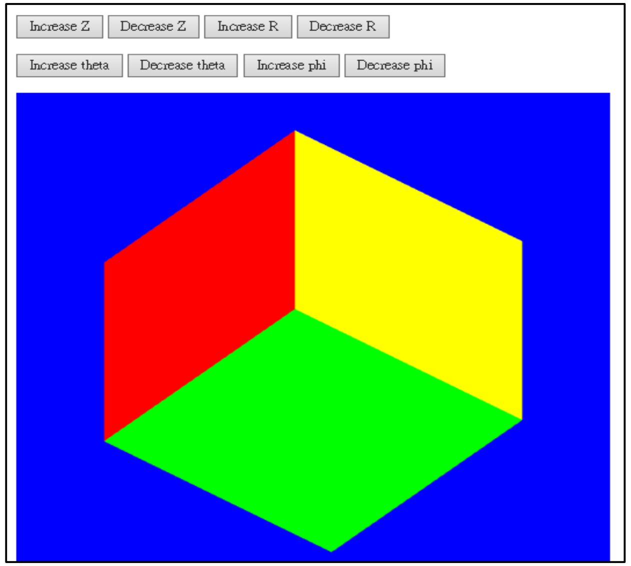
ortho.js (8/10)

```
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 );

modelView = gl.getUniformLocation( program, "modelView" );
projection   = gl.getUniformLocation( program, "projection" );
```



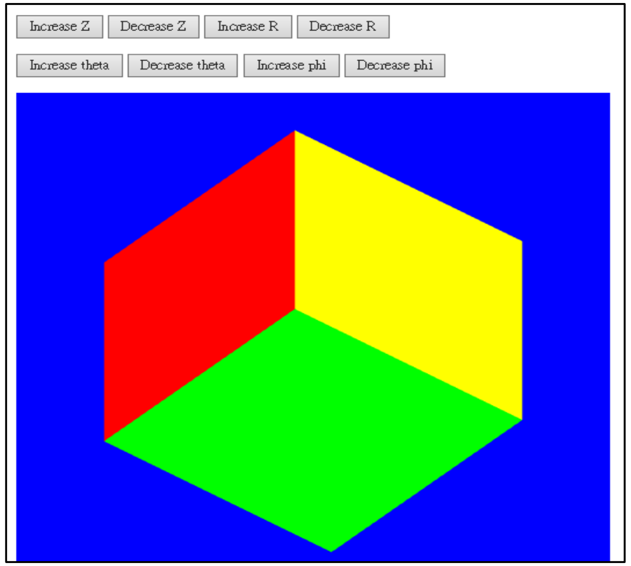
ortho.js (9/10)

```
// buttons to change viewing parameters

    document.getElementById("Button1").onclick = function() {near  *= 1.1; far *= 1.1;};
    document.getElementById("Button2").onclick = function() {near *= 0.9; far *= 0.9;};
    document.getElementById("Button3").onclick = function() {radius *= 1.1;};
    document.getElementById("Button4").onclick = function() {radius *= 0.9;};
    document.getElementById("Button5").onclick = function() {theta += dr;};
    document.getElementById("Button6").onclick = function() {theta -= dr;};
    document.getElementById("Button7").onclick = function() {phi += dr;};
    document.getElementById("Button8").onclick = function() {phi -= dr;};

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

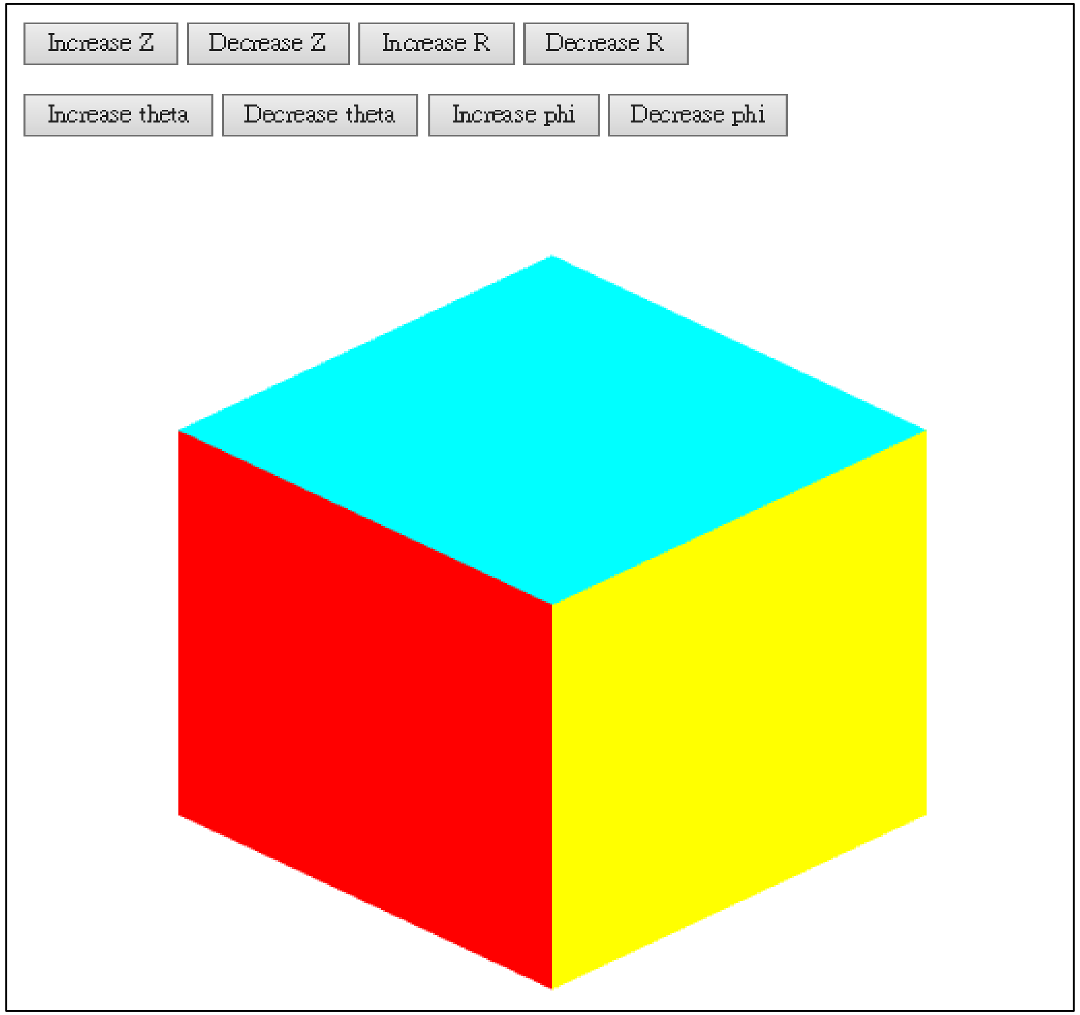


ortho.js (10/10)

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

        eye = vec3(radius*Math.sin(phi), radius*Math.sin(theta),
                          radius*Math.cos(phi));

        mvMatrix = lookAt(eye, at , up);
        pMatrix    = ortho(left, right, bottom, ytop, near, far);

        gl.uniformMatrix4fv( modelView, false, flatten(mvMatrix) );
        gl.uniformMatrix4fv( projection,   false, flatten(pMatrix) );

        gl.drawArrays( gl.TRIANGLES, 0, numVertices );
        requestAnimFrame(render);
    }  // end of render()
```



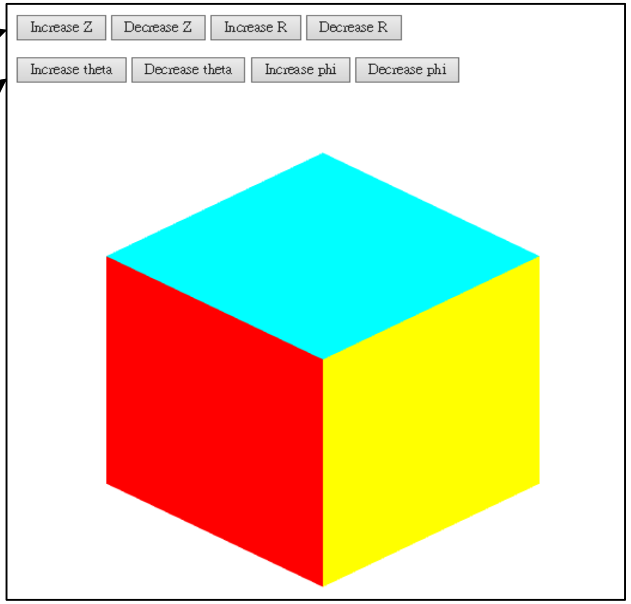
Sample Programs: ortho1.html, ortho1.js



Interactive orthographic viewing of cube

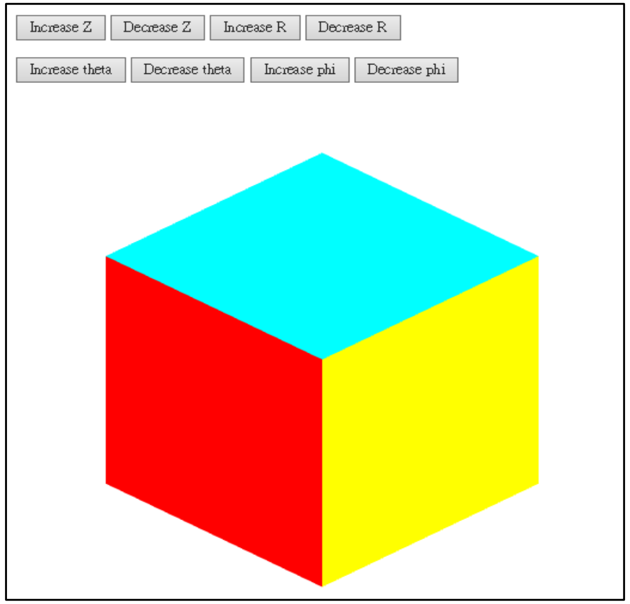
ortho1.html (1/4)

```
<!DOCTYPE html>
<html>


<p> </p>
<button id = "Button1">Increase Z</button>
<button id = "Button2">Decrease Z</button>
<button id = "Button3">Increase R</button>
<button id = "Button4">Decrease R</button>

<p> </p>
<button id = "Button5">Increase theta</button>
<button id = "Button6">Decrease theta</button>
<button id = "Button7">Increase phi</button>
<button id = "Button8">Decrease phi</button>
<p> </p>
```



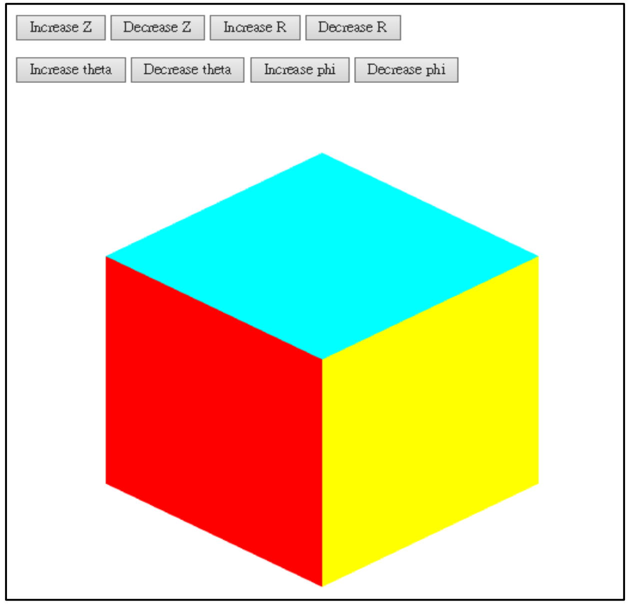
ortho1.html (2/4)

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

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

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

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



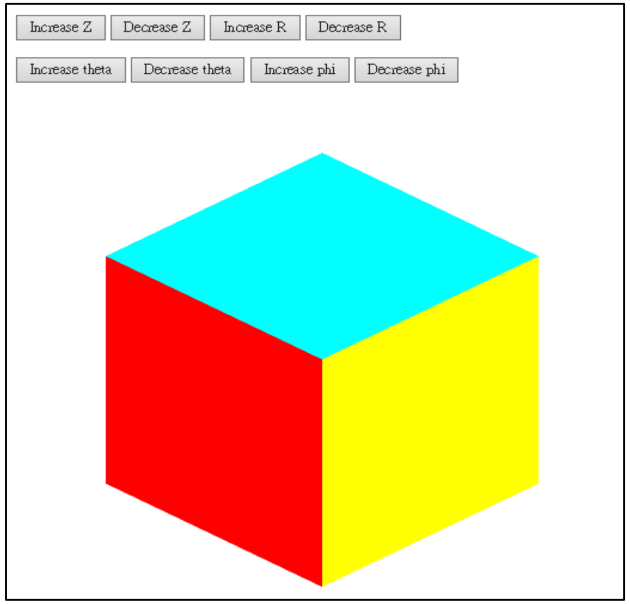
ortho1.html (3/4)

```
<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="ortho1.js"></script>
```



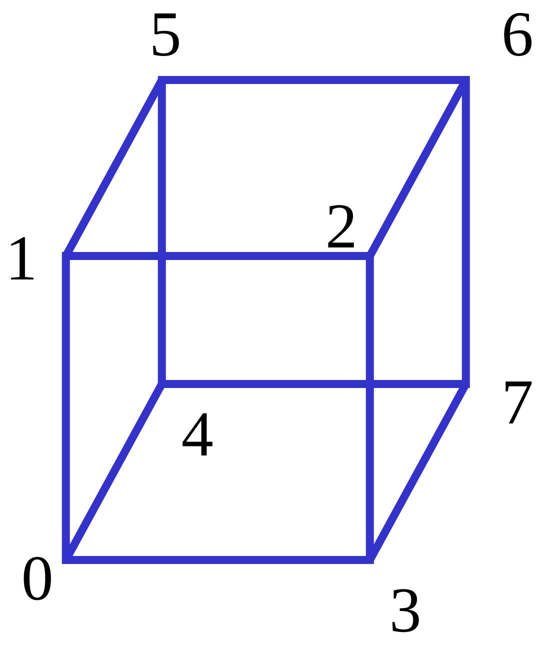
ortho1.html (4/4)

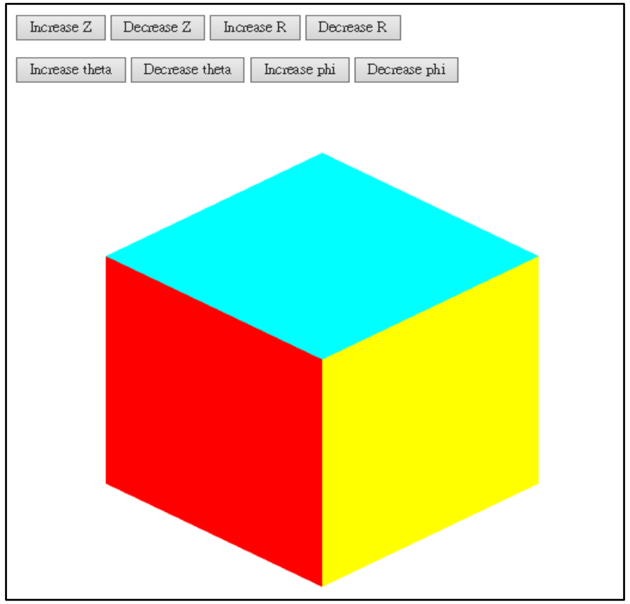
```
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>

</body>
</html>
```



ortho1.js (1/10)

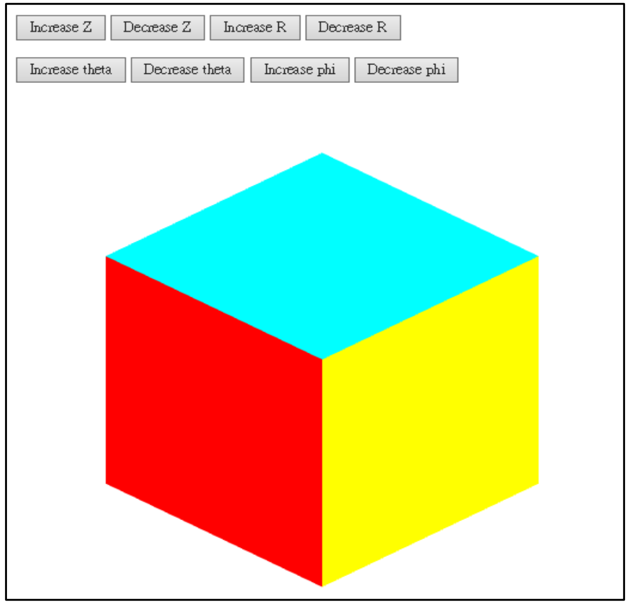
```
var canvas;
var gl;

var numVertices  = 36;

var pointsArray = [];
var colorsArray = [];

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 ),
    ];
```





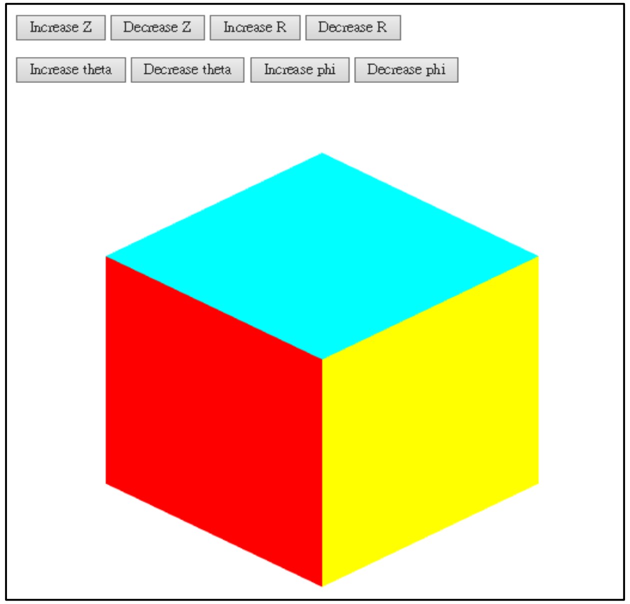
ortho1.js (2/10)

```
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
    ];
```



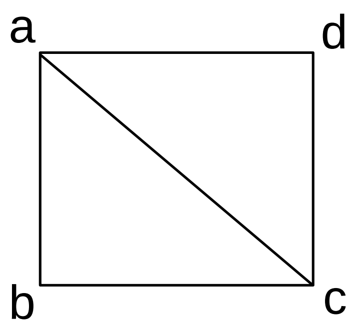
ortho1.js (3/10)

```
var near = -1;
var far = 1;
var radius = 1.0;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;

var left = -1.0;
var right = 1.0;
var ytop = 1.0;
var bottom = -1.0;

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var eye;

const at  = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);
```

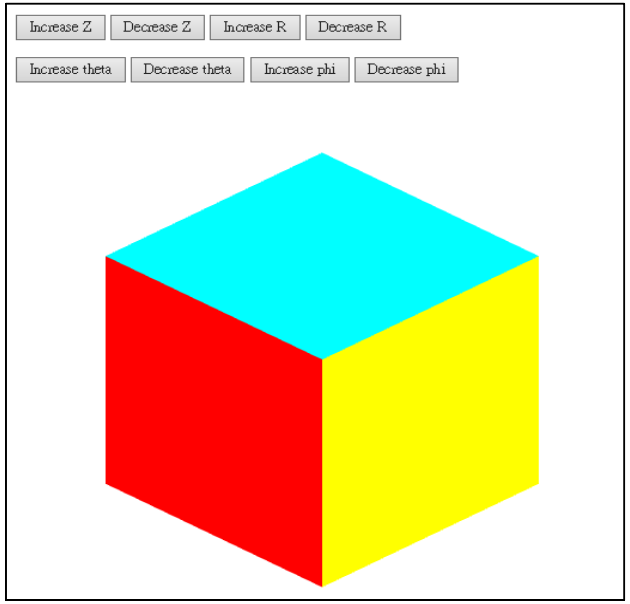


ortho1.js (4/10)

```
// quad uses first index to set color for face

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

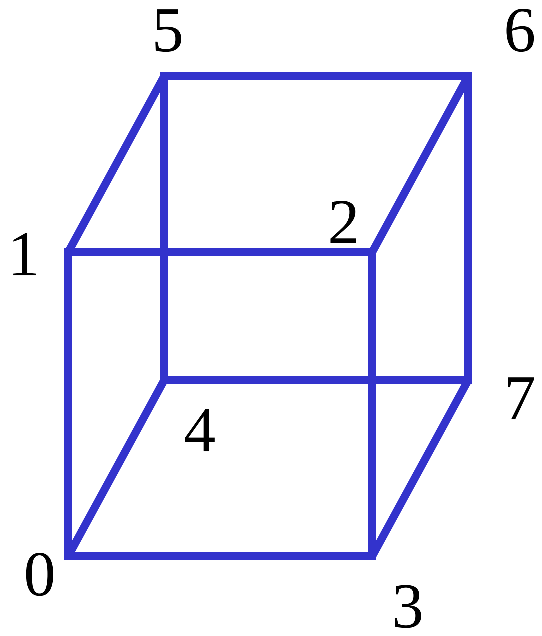


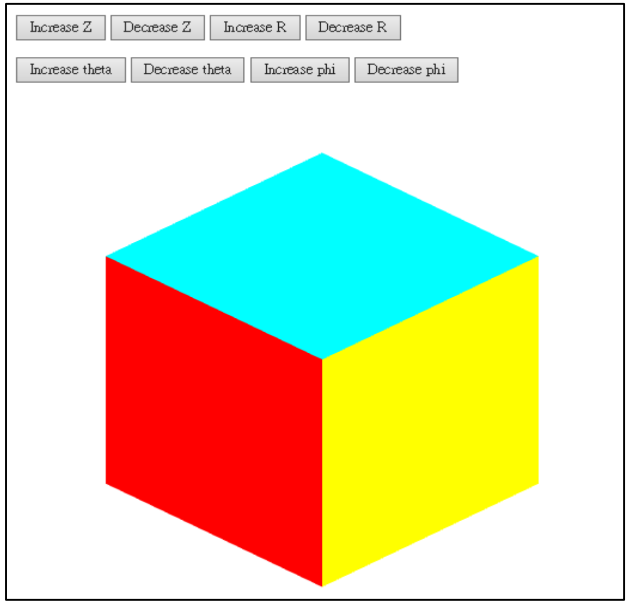
gl.TRIANGLES



ortho1.js (5/10)

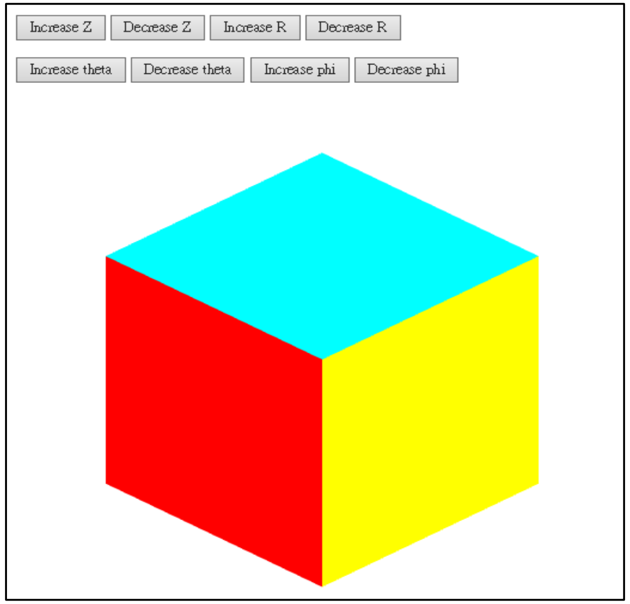
```
// Each face determines two triangles

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 );
}
```





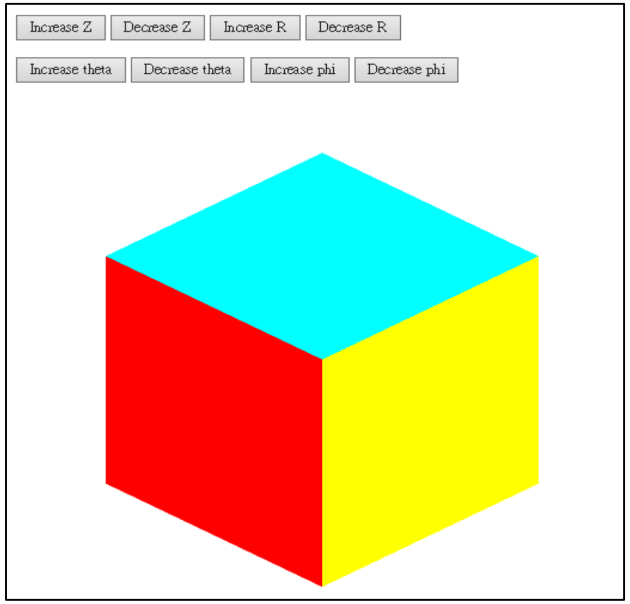
ortho1.js (6/10)

```
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);
```



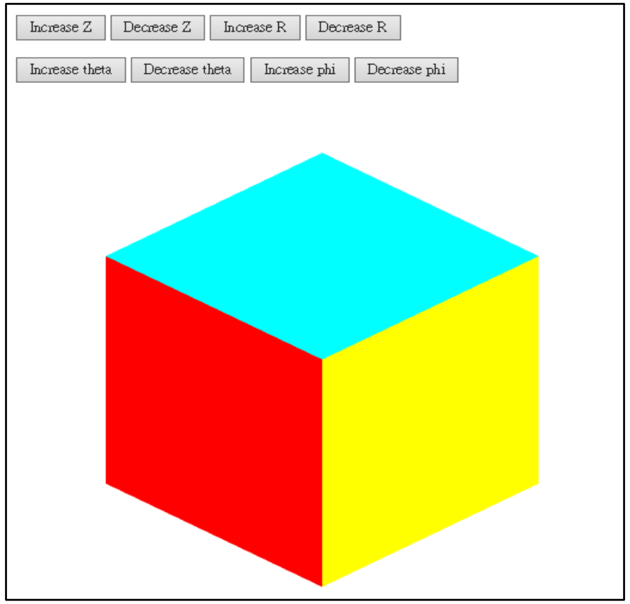
ortho1.js (7/10)

```
//
//  Load shaders and initialize attribute buffers
//
var 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);
```



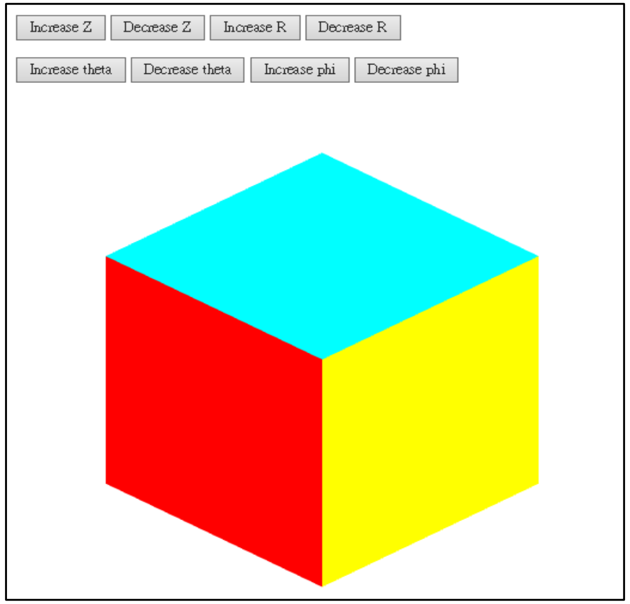
ortho1.js (8/10)

```
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 );

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



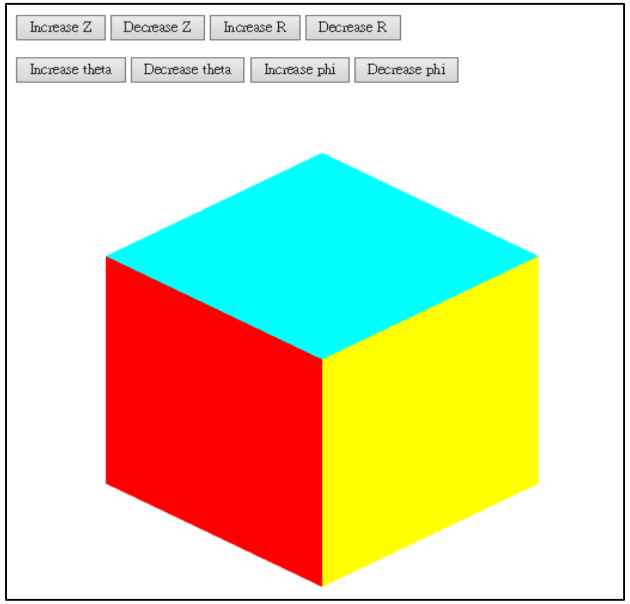
ortho1.js (9/10)

```
// buttons to change viewing parameters

   document.getElementById("Button1").onclick = function() {near  *= 1.1; far *= 1.1;};
   document.getElementById("Button2").onclick = function() {near *= 0.9; far *= 0.9;};
   document.getElementById("Button3").onclick = function() {radius *= 1.1;};
   document.getElementById("Button4").onclick = function() {radius *= 0.9;};
   document.getElementById("Button5").onclick = function() {theta += dr;};
   document.getElementById("Button6").onclick = function() {theta -= dr;};
   document.getElementById("Button7").onclick = function() {phi += dr;};
   document.getElementById("Button8").onclick = function() {phi -= dr;};

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

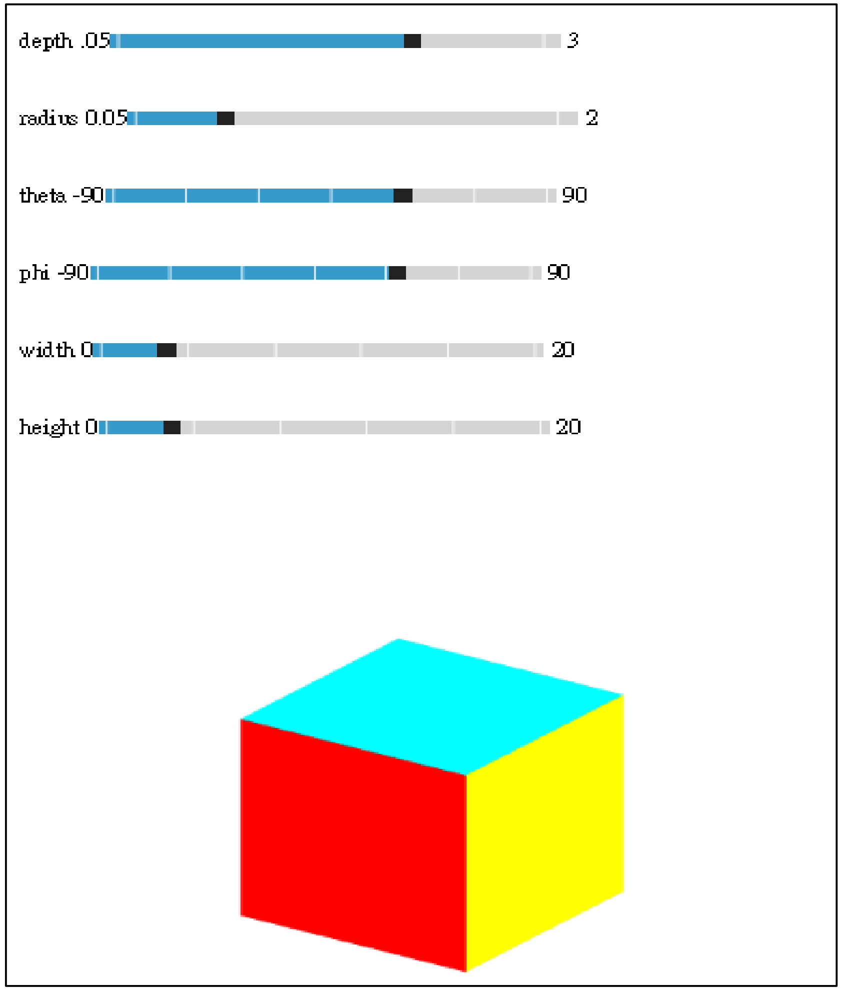


ortho1.js (10/10)

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

        eye = vec3(radius*Math.sin(phi), radius*Math.sin(theta),
                          radius*Math.cos(phi));

        modelViewMatrix = lookAt(eye, at , up);
        projectionMatrix   = ortho(left, right, bottom, ytop, 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()
```



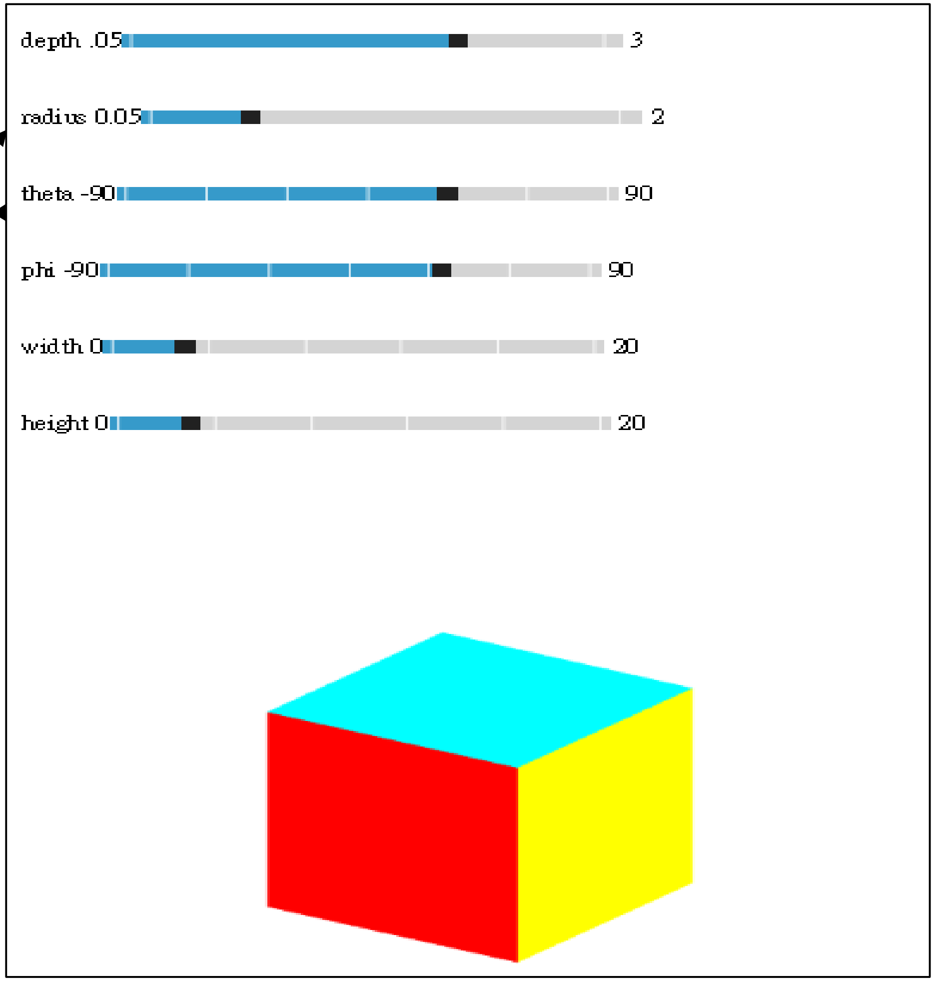
Sample Programs: ortho2.html, ortho2.js



Use slide bars instead of buttons

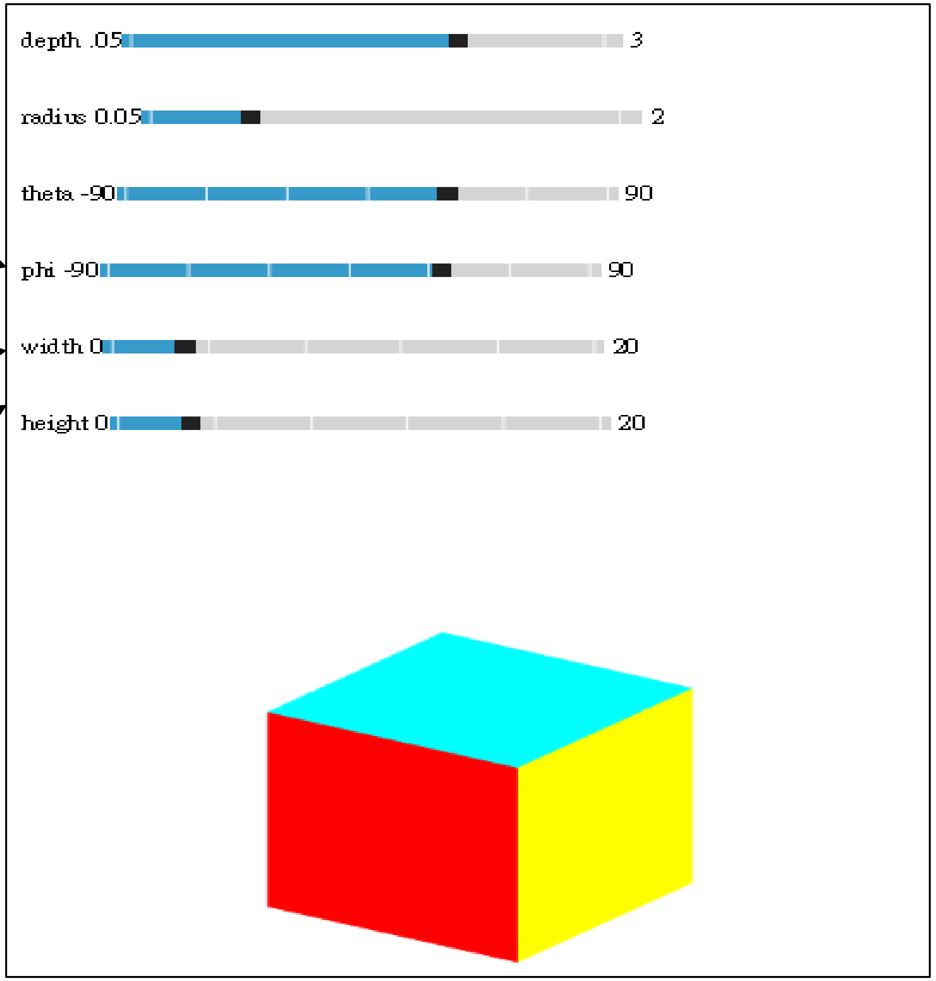
ortho2.html (1/5)

```
<!DOCTYPE html>
<html>

<div>
depth .05<input id="depthSlider" type="range"
 min=".05" max="3" step="0.1" value="2" />
 3
</div>
<div>
radius 0.05<input id="radiusSlider" type="range"
 min="0.05" max="2" step="0.1" value="1" />
 2
</div>
<div>
theta -90<input id="thetaSlider" type="range"
 min="-90" max="90" step="5" value="0" />
 90
</div>
```



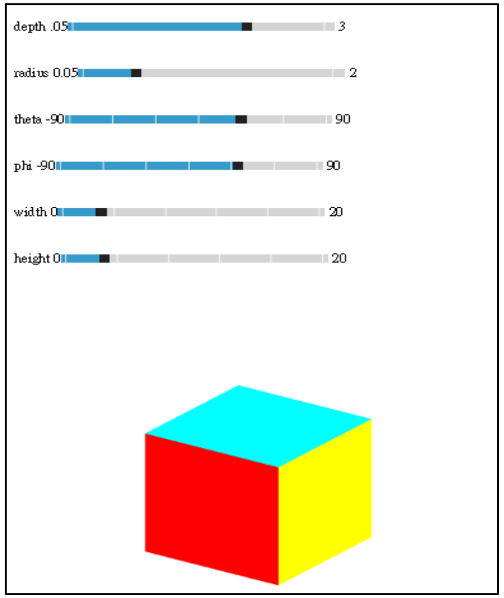
ortho2.html (2/5)

```
<div>
phi -90<input id="phiSlider" type="range"
 min="-90" max="90" step="5" value="0" />
 90
</div>
<div>
width 0<input id="widthSlider" type="range"
 min="0" max="20" step="1" value="10" />
 20
</div>
<div>
height 0<input id="heightSlider" type="range"
 min="0" max="20" step="1" value="10" />
 20
</div>
```



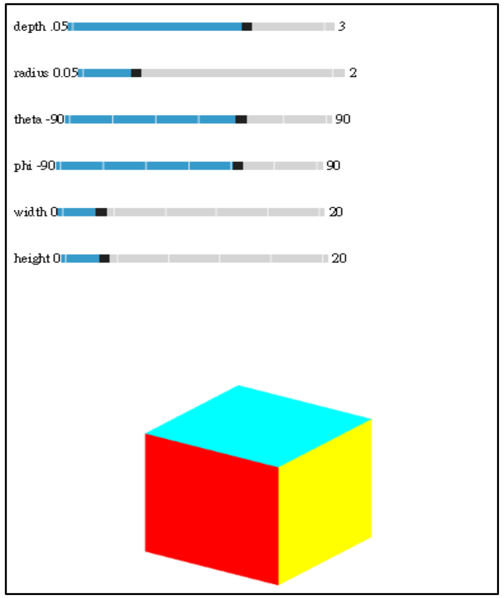
ortho2.html (3/5)

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

attribute  vec4 vPosition;
attribute  vec4 vColor;
varying vec4 fcolor;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

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



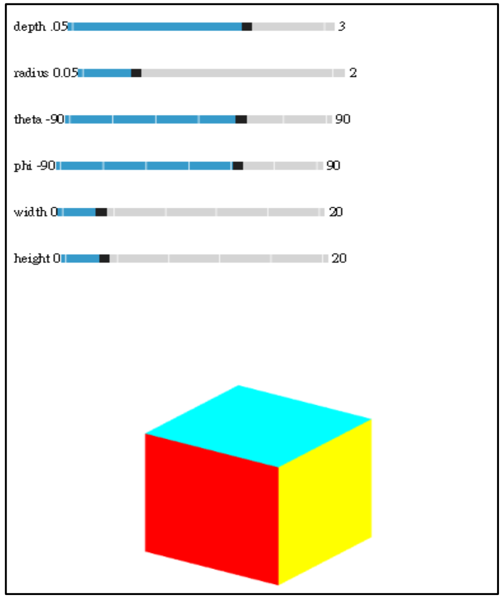
ortho2.html (4/5)

```
<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="ortho2.js"></script>
```



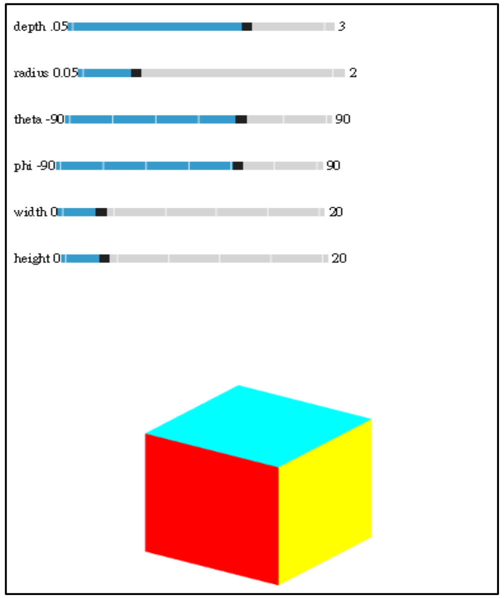
ortho2.html (5/5)

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

</body>
</html>
```



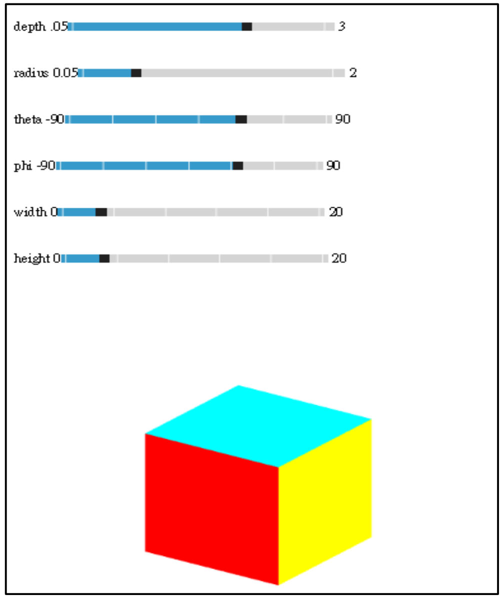
ortho2.js (1/11)

```
var canvas;
var gl;

var numVertices  = 36;

var pointsArray = [];
var colorsArray = [];

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 ),
    ];
```



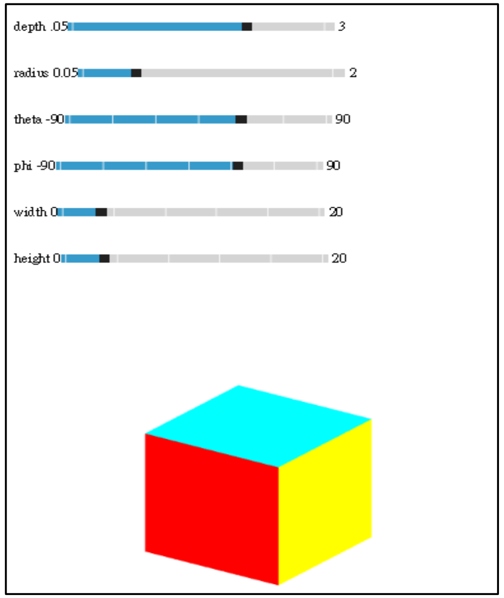
ortho2.js (2/11)

```
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
    ];
```



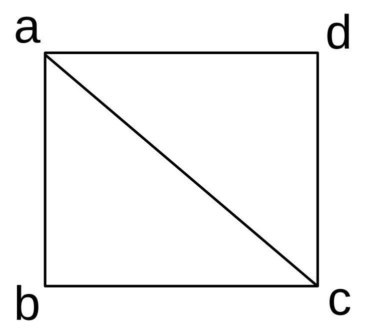
ortho2.js (3/11)

```
var near = -1;
var far = 1;
var radius = 1;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;

var left = -1.0;
var right = 1.0;
var ytop = 1.0;
var bottom = -1.0;

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var eye;
const at  = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);
```

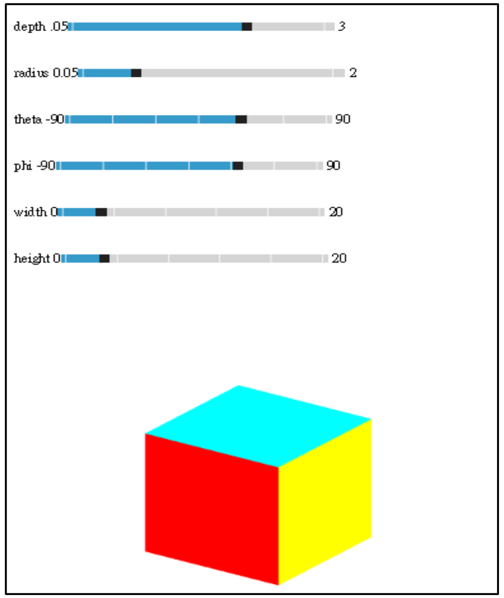


ortho2.js (4/11)

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

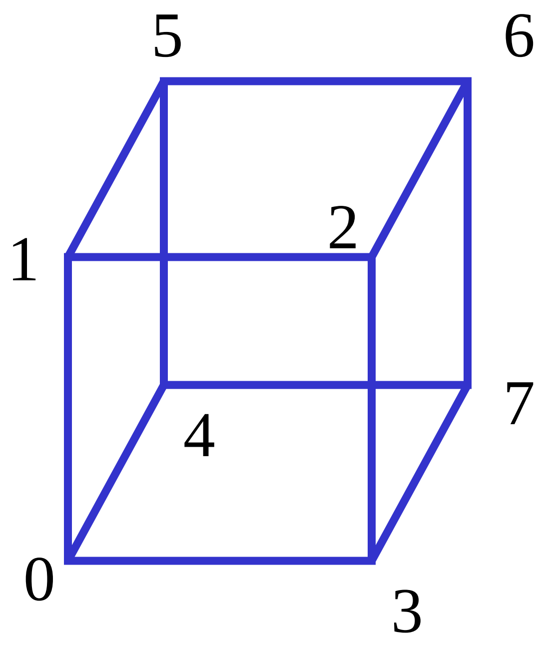


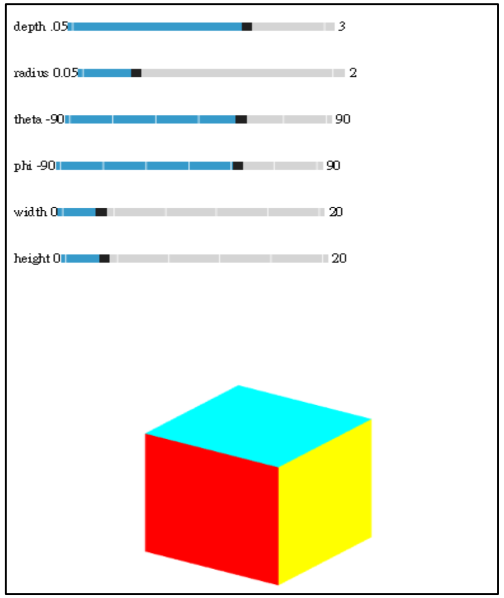
gl.TRIANGLES



ortho2.js (5/11)

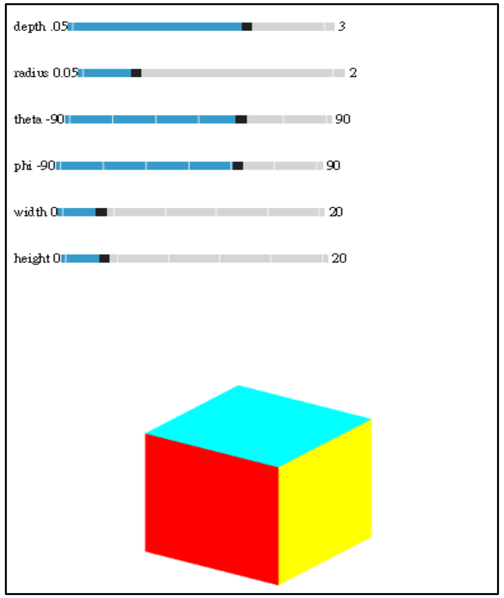
```
function colorCube()
{
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```





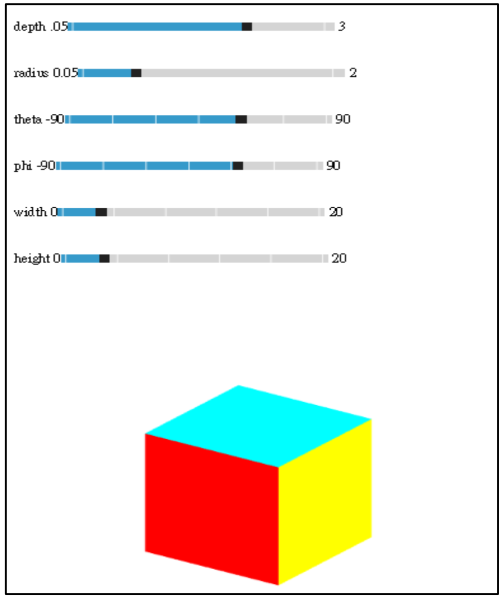
ortho2.js (6/11)

```
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);
```



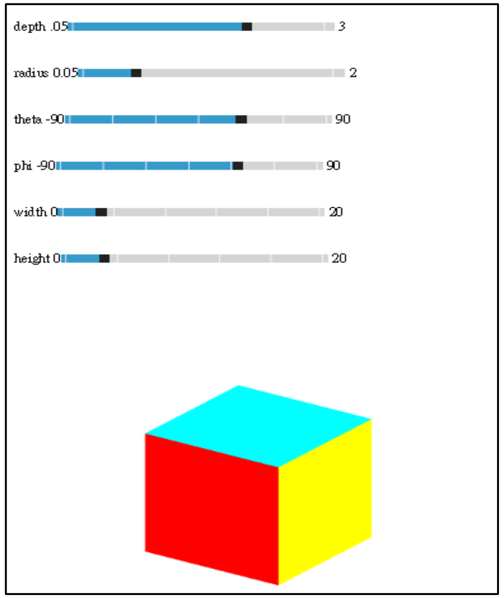
ortho2.js (7/11)

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

colorCube();

var cBufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBufferId );
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);
```



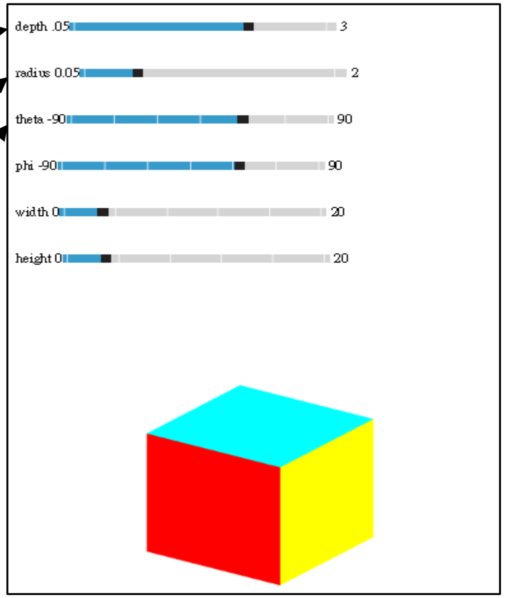
ortho2.js (8/11)

```
var vBufferId = gl.createBuffer();
  gl.bindBuffer( gl.ARRAY_BUFFER, vBufferId );
  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 );

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



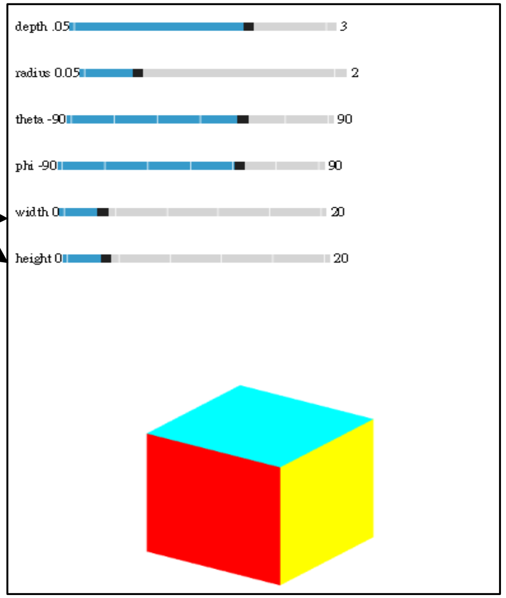
ortho2.js (9/11)

```
// sliders for viewing parameters

    document.getElementById("depthSlider").onchange = function() {
        far = event.srcElement.value/2;
        near = -event.srcElement.value/2;
    };

    document.getElementById("radiusSlider").onchange = function() {
       radius = event.srcElement.value;
    };

    document.getElementById("thetaSlider").onchange = function() {
        theta = event.srcElement.value* Math.PI/180.0;
    };

  document.getElementById("phiSlider").onchange = function() {
        phi = event.srcElement.value* Math.PI/180.0;
    };
```



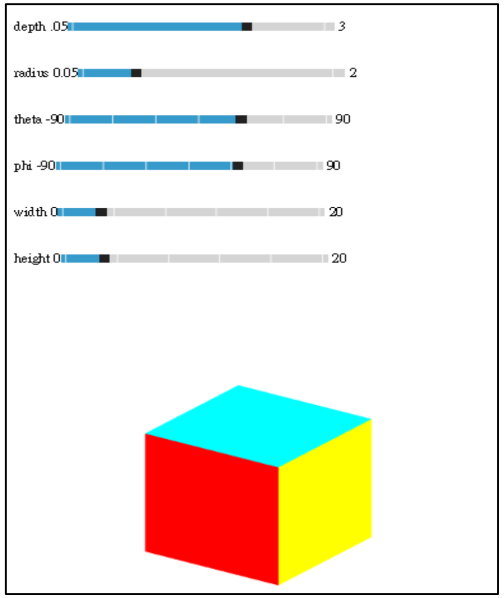
ortho2.js (10/11)

```
    document.getElementById("heightSlider").onchange = function() {
        ytop = event.srcElement.value/2;
        bottom = -event.srcElement.value/2;
    };
    document.getElementById("widthSlider").onchange = function() {
        right = event.srcElement.value/2;
        left = -event.srcElement.value/2;
    };

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

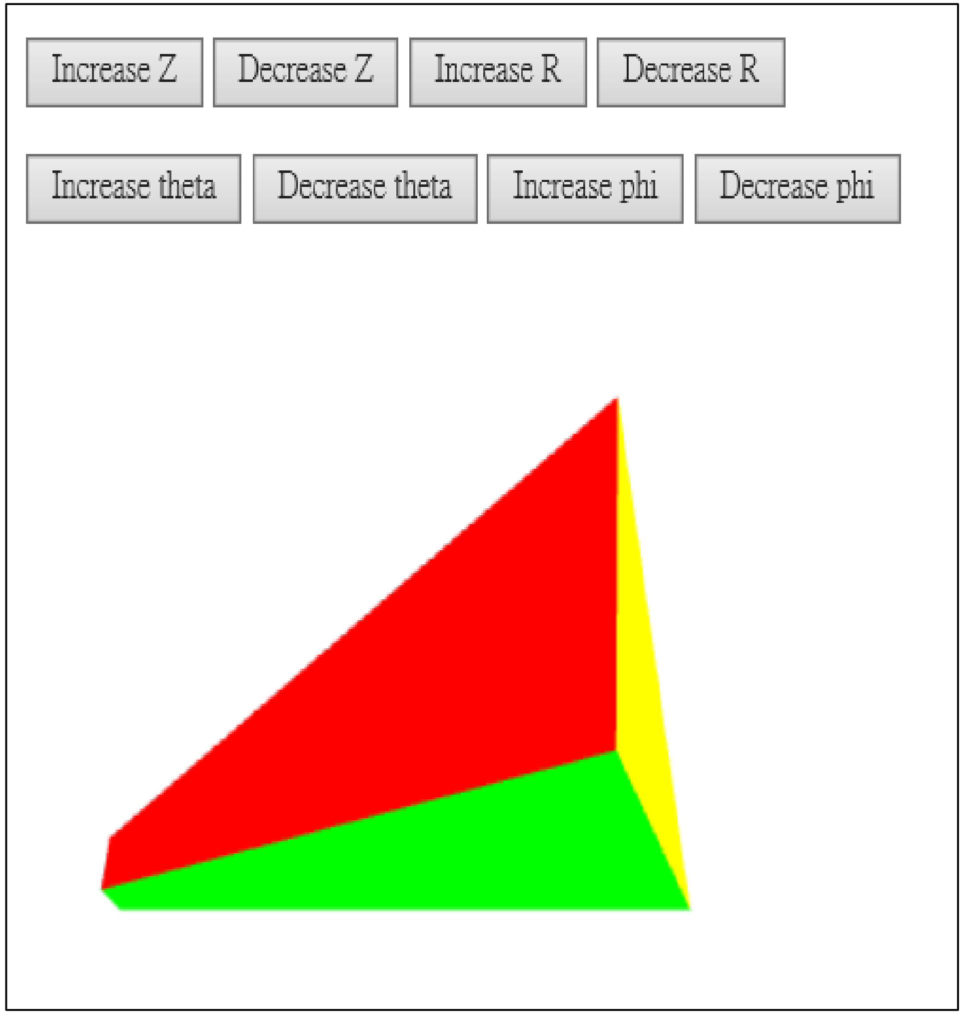


ortho2.js (11/11)

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

        eye = vec3(radius*Math.sin(phi), radius*Math.sin(theta),
                          radius*Math.cos(phi));

        modelViewMatrix = lookAt(eye, at , up);
        projectionMatrix  = ortho(left, right, bottom, ytop, near, far);

        gl.uniformMatrix4fv( modelViewMatrixLoc, false, flatten(modelViewMatrix) );
        gl.uniformMatrix4fv( projectionMatrixLoc,  false, flatten(projectionMatrix) );

        gl.drawArrays( gl.TRIANGLES, 0, numVertices );
        requestAnimFrame(render);
    }
```



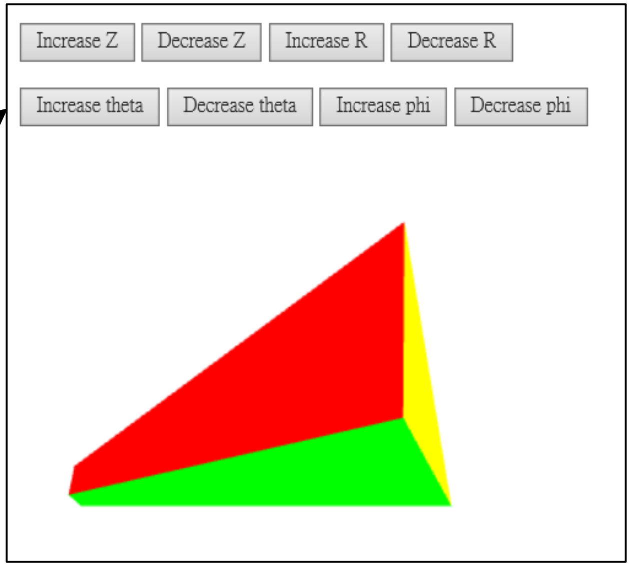
Sample Programs: perspective.html, perspective.js



Interactive perspective viewing of cube

perspective.html (1/4)

```
!DOCTYPE html>
<html>

<p> </p>
<button id = "Button1">Increase Z</button>
<button id = "Button2">Decrease Z</button>
<button id = "Button3">Increase R</button>
<button id = "Button4">Decrease R</button>

<p> </p>
<button id = "Button5">Increase theta</button>
<button id = "Button6">Decrease theta</button>
<button id = "Button7">Increase phi</button>
<button id = "Button8">Decrease phi</button>
<p> </p>
```



perspective.html (2/4)

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

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

uniform mat4 modelView;
uniform mat4 projection;

void main()
{
    gl_Position = projection*modelView*vPosition;
    fColor = vColor;
}
</script>
```

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

perspective.html (4/4)

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

</body>
</html>
```

perspective.js (1/10)

```
var canvas;
var gl;

var NumVertices  = 36;

var pointsArray = [];
var colorsArray = [];

var vertices = [
    vec4(-0.5, -0.5,  1.5, 1.0),
    vec4(-0.5,  0.5,  1.5, 1.0),
    vec4( 0.5,  0.5,  1.5, 1.0),
    vec4( 0.5, -0.5,  1.5, 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)
];
```

perspective.js (2/10)

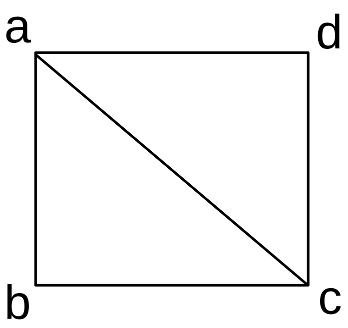
```
var vertexColors = [
    vec4( 0.0, 0.0, 0.0, 1.0 ),  // black
    vec4( 1.0, 0.0, 0.0, 1.0 ),  // red
    vec4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    vec4( 0.0, 1.0, 0.0, 1.0 ),  // green
    vec4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    vec4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
    vec4( 0.0, 1.0, 1.0,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

perspective.js (3/10)

```
var near = 0.3;
var far = 3.0;
var radius = 4.0;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;

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

var mvMatrix, pMatrix;
var modelView, projection;
var eye;
const at  = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);
```

perspective.js (4/10)

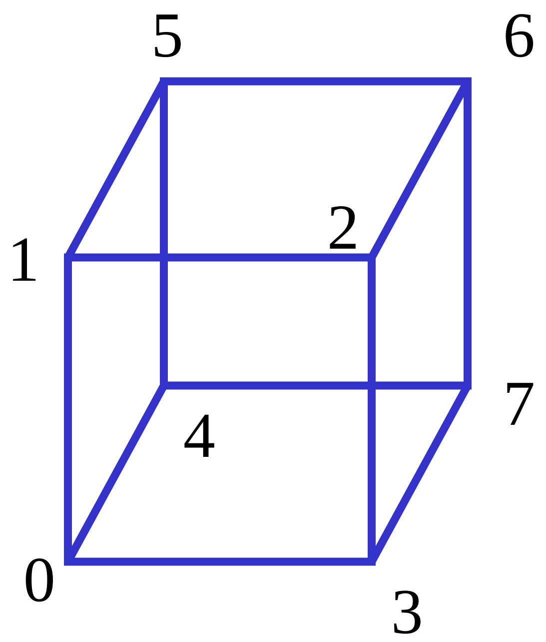
```
function quad(a, b, c, d) {
    pointsArray.push(vertices[a]);
    colorsArray.push(vertexColors[a]);
    pointsArray.push(vertices[b]);
    colorsArray.push(vertexColors[a]);
    pointsArray.push(vertices[c]);
    colorsArray.push(vertexColors[a]);
    pointsArray.push(vertices[a]);
    colorsArray.push(vertexColors[a]);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



gl.TRIANGLES

perspective.js (5/10)

```
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 );
}
```



perspective.js (6/10)

```
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 );

    aspect =  canvas.width/canvas.height;

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

    gl.enable(gl.DEPTH_TEST);
```

perspective.js (7/10)

```
//
//  Load shaders and initialize attribute buffers
//
var 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);
```

perspective.js (8/10)

```
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 );

modelView = gl.getUniformLocation( program, "modelView" );
projection   = gl.getUniformLocation( program, "projection" );
```

perspective.js (9/10)

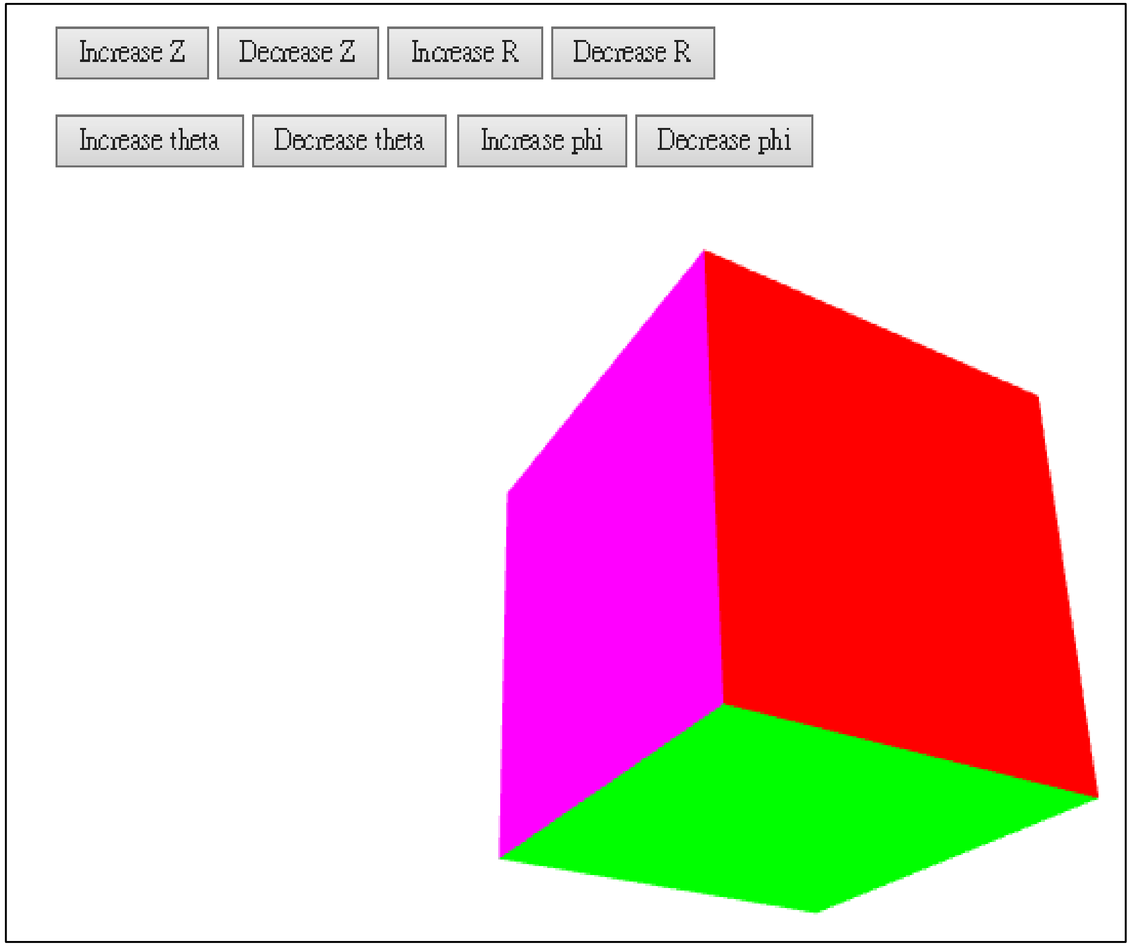
```
// buttons for viewing parameters

    document.getElementById("Button1").onclick = function() {near  *= 1.1; far *= 1.1;};
    document.getElementById("Button2").onclick = function() {near *= 0.9; far *= 0.9;};
    document.getElementById("Button3").onclick = function() {radius *= 2.0;};
    document.getElementById("Button4").onclick = function() {radius *= 0.5;};
    document.getElementById("Button5").onclick = function() {theta += dr;};
    document.getElementById("Button6").onclick = function() {theta -= dr;};
    document.getElementById("Button7").onclick = function() {phi += dr;};
    document.getElementById("Button8").onclick = function() {phi -= dr;};

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

perspective.js (10/10)

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

    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
               radius*Math.sin(theta)*Math.sin(phi),
               radius*Math.cos(theta));
    mvMatrix = lookAt(eye, at , up);
    pMatrix    = perspective(fovy, aspect, near, far);

    gl.uniformMatrix4fv( modelView, false, flatten(mvMatrix) );
    gl.uniformMatrix4fv( projection,  false,  flatten(pMatrix) );

    gl.drawArrays( gl.TRIANGLES, 0, NumVertices );
    requestAnimFrame(render);
}   // end of render()
```

$$\mathrm{eye}(r, \theta, \emptyset) \equiv \mathrm{eye}(x, y, z) \begin{cases} x = r \sin\theta \cos\emptyset \\ y = r \sin\theta \sin\emptyset \\ z = r\cos\theta \end{cases}$$

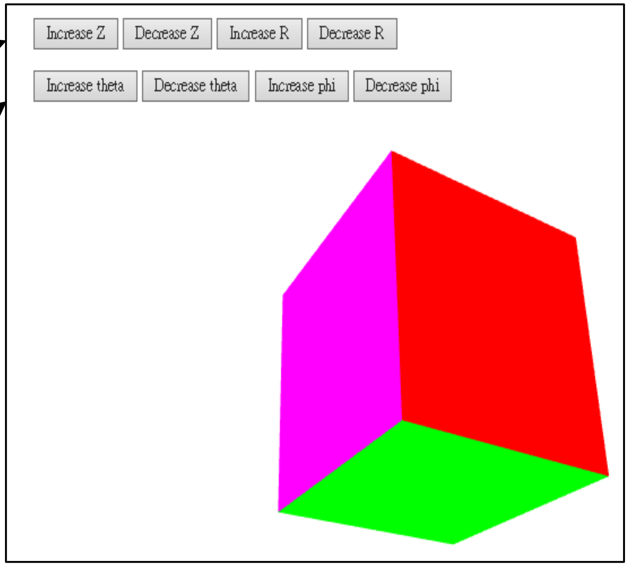
Sample Programs: perspective1.html, perspective1.js



Interactive perspective viewing of cube

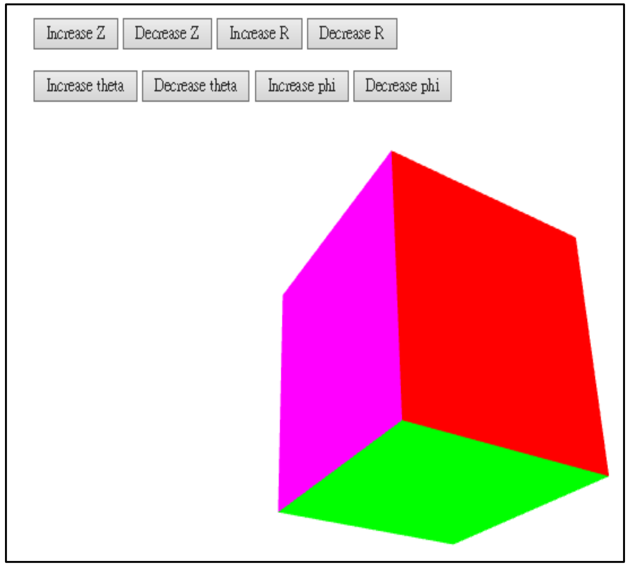
perspective1.html (1/4)

```
<!DOCTYPE html>
<html>

<p> </p>
<button id = "Button1">Increase Z</button>
<button id = "Button2">Decrease Z</button>
<button id = "Button3">Increase R</button>
<button id = "Button4">Decrease R</button>

<p> </p>
<button id = "Button5">Increase theta</button>
<button id = "Button6">Decrease theta</button>
<button id = "Button7">Increase phi</button>
<button id = "Button8">Decrease phi</button>
<p> </p>
```



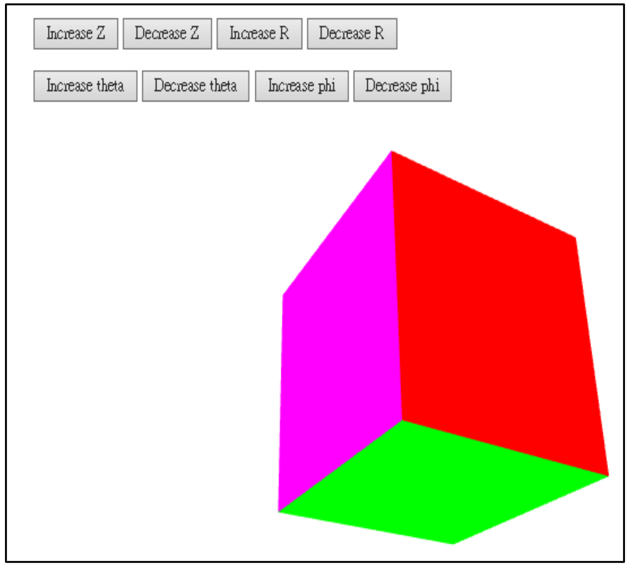
perspective1.html (2/4)

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

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

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

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



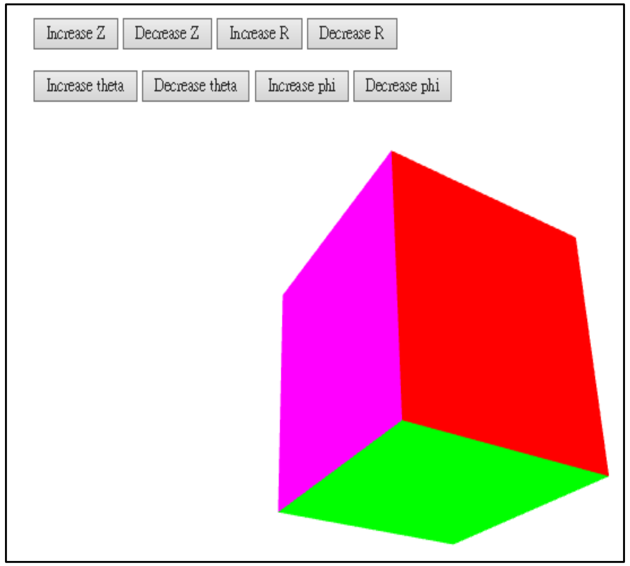
perspective1.html (3/4)

```
<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="perspective1.js"></script>
```



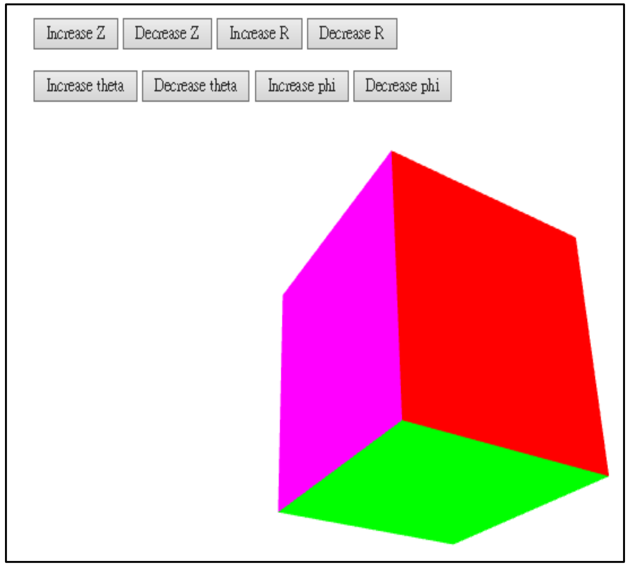
perspective1.html (4/4)

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

</body>
</html>
```



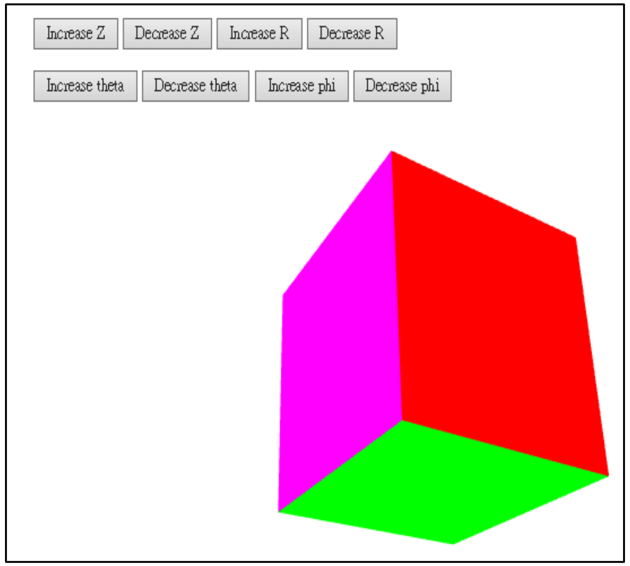
perspective1.js (1/10)

```
var canvas;
var gl;

var NumVertices  = 36;

var pointsArray = [];
var colorsArray = [];

var vertices = [
    vec4(-0.5, -0.5,  1.5, 1.0),
    vec4(-0.5,  0.5,  1.5, 1.0),
    vec4( 0.5,  0.5,  1.5, 1.0),
    vec4( 0.5, -0.5,  1.5, 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)
];
```



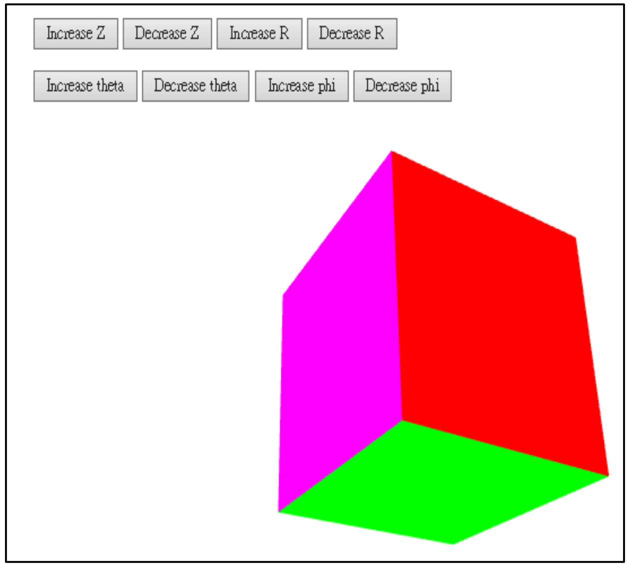
perspective1.js (2/10)

```
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
];
```



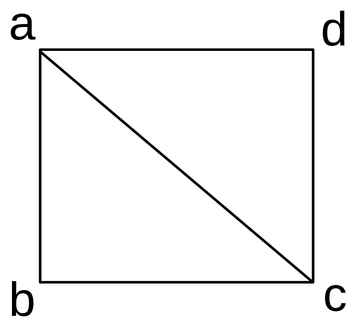
perspective1.js (3/10)

```
var near = 0.3;
var far = 3.0;
var radius = 4.0;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;

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

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var eye;
const at = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);
```

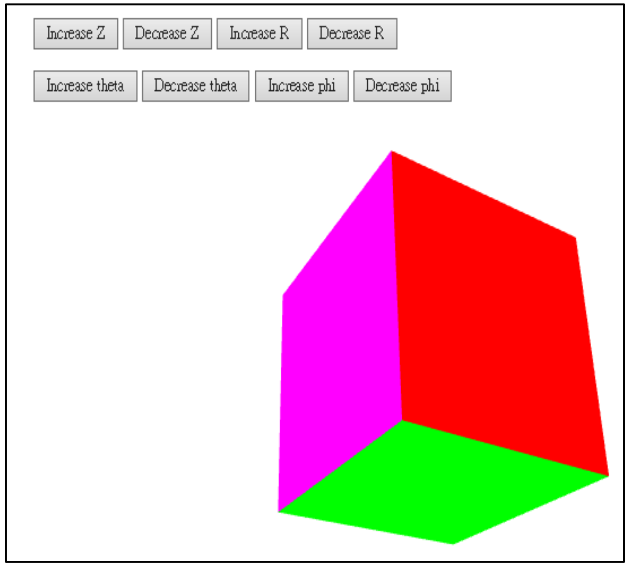


perspective1.js (4/10)

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

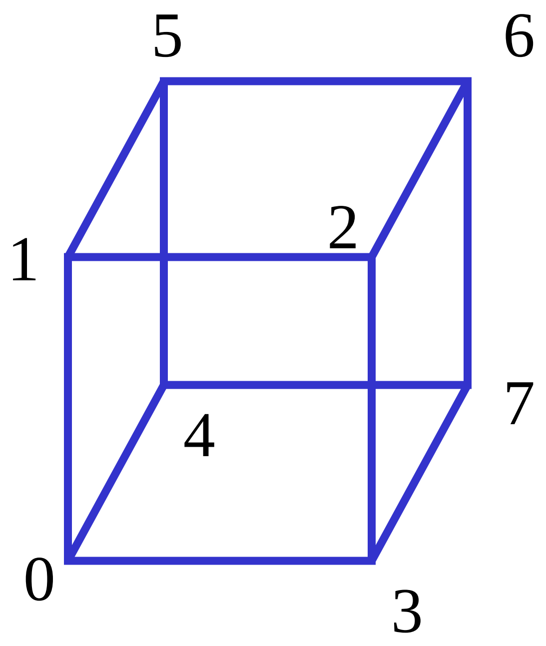


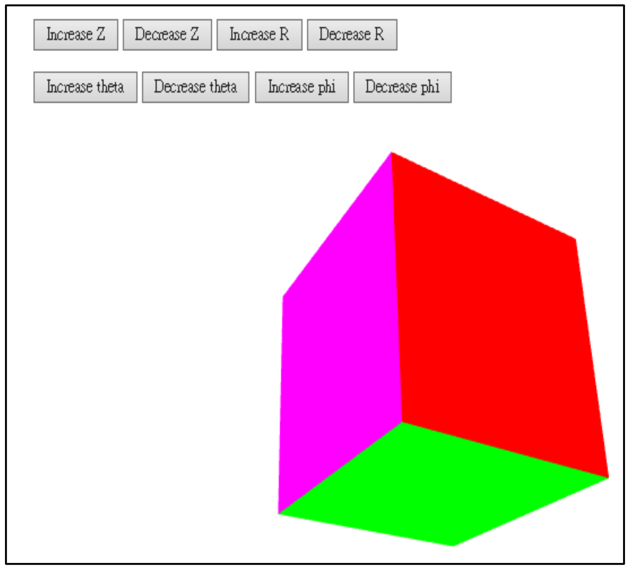
gl.TRIANGLES



perspective1.js (5/10)

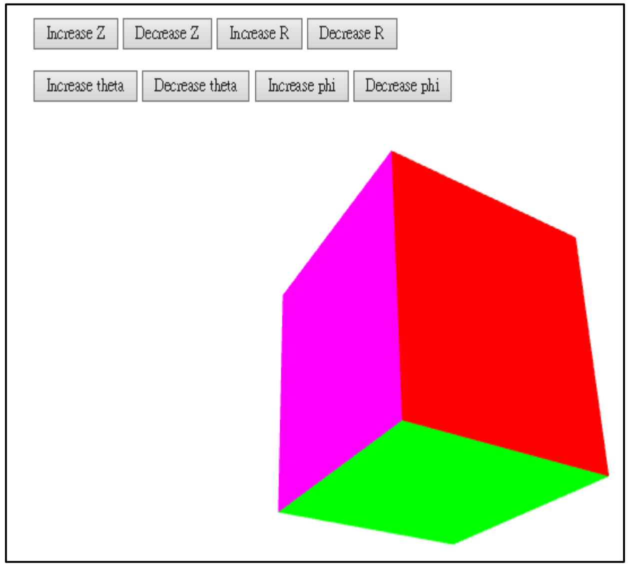
```
function colorCube()
{
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```





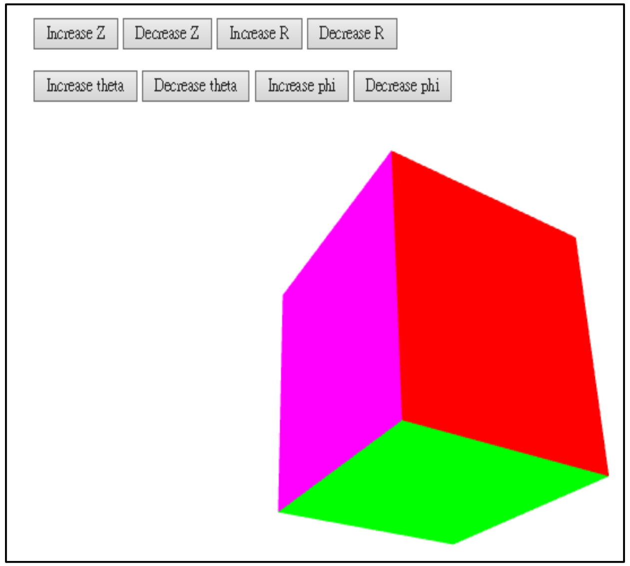
perspective1.js (6/10)

```
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 );

    aspect =  canvas.width/canvas.height;

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

    gl.enable(gl.DEPTH_TEST);
```



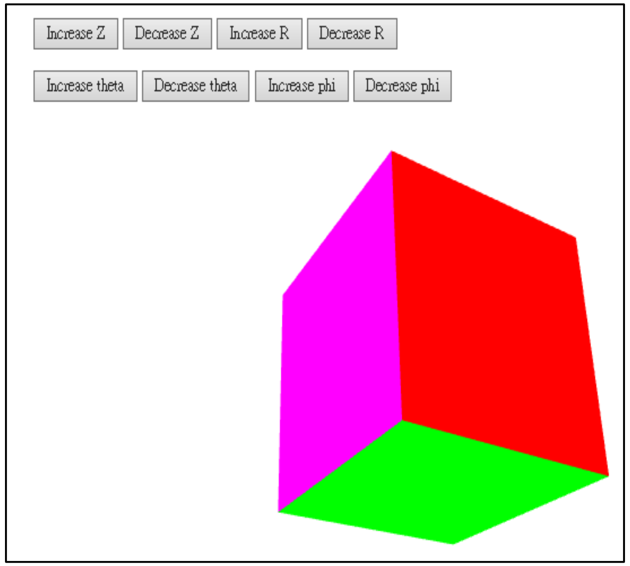
perspective1.js (7/10)

```
//
//  Load shaders and initialize attribute buffers
//
var 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);
```



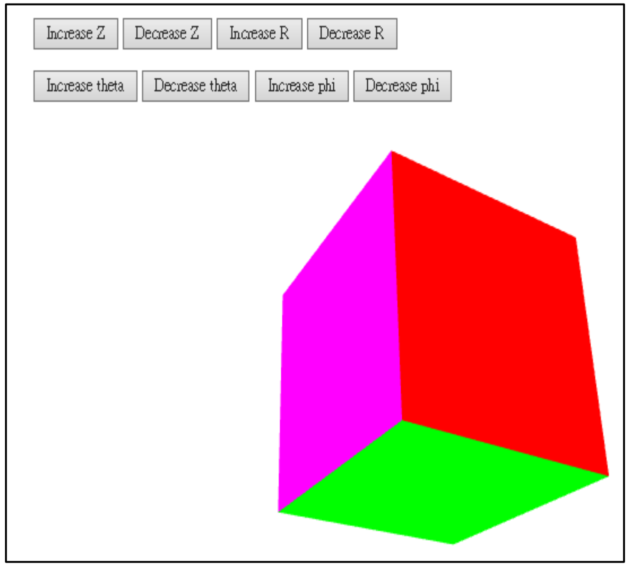
perspective1.js (8/10)

```
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 );

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



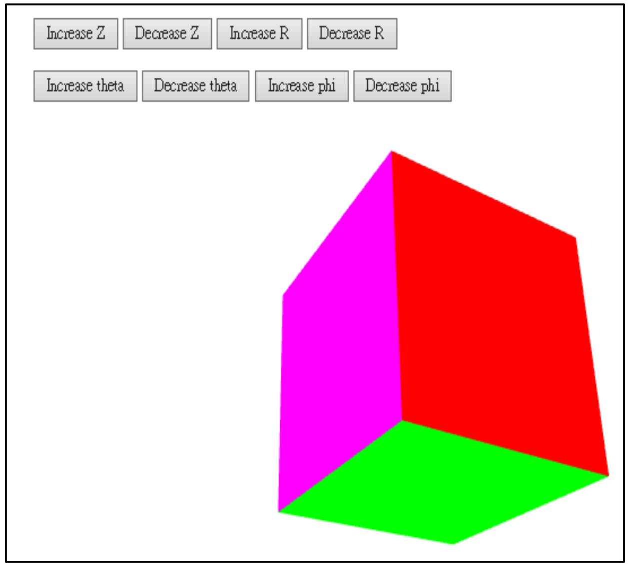
perspective1.js (9/10)

```
// buttons for viewing parameters

    document.getElementById("Button1").onclick = function() {near  *= 1.1; far *= 1.1;};
    document.getElementById("Button2").onclick = function() {near *= 0.9; far *= 0.9;};
    document.getElementById("Button3").onclick = function() {radius *= 2.0;};
    document.getElementById("Button4").onclick = function() {radius *= 0.5;};
    document.getElementById("Button5").onclick = function() {theta += dr;};
    document.getElementById("Button6").onclick = function() {theta -= dr;};
    document.getElementById("Button7").onclick = function() {phi += dr;};
    document.getElementById("Button8").onclick = function() {phi -= dr;};

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

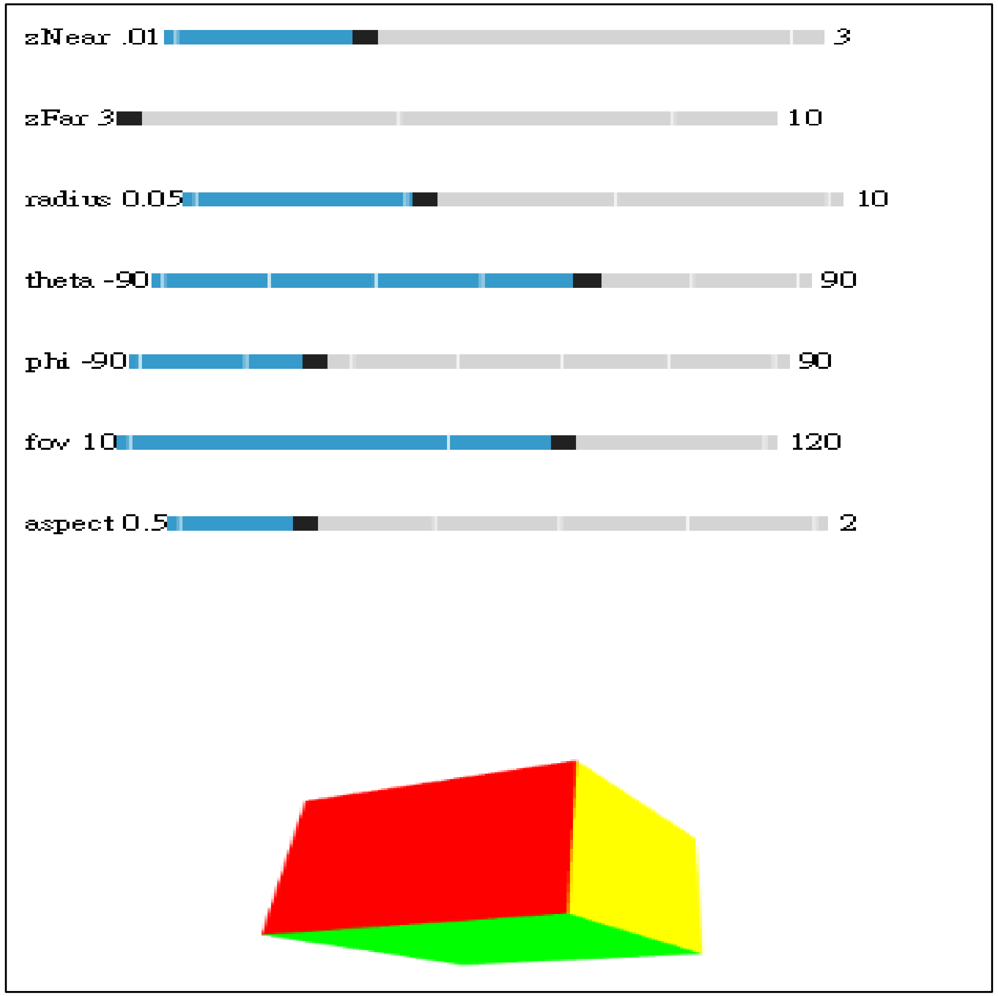


perspective1.js (10/10)

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

    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
                    radius*Math.sin(theta)*Math.sin(phi),
                    radius*Math.cos(theta));
    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()
```



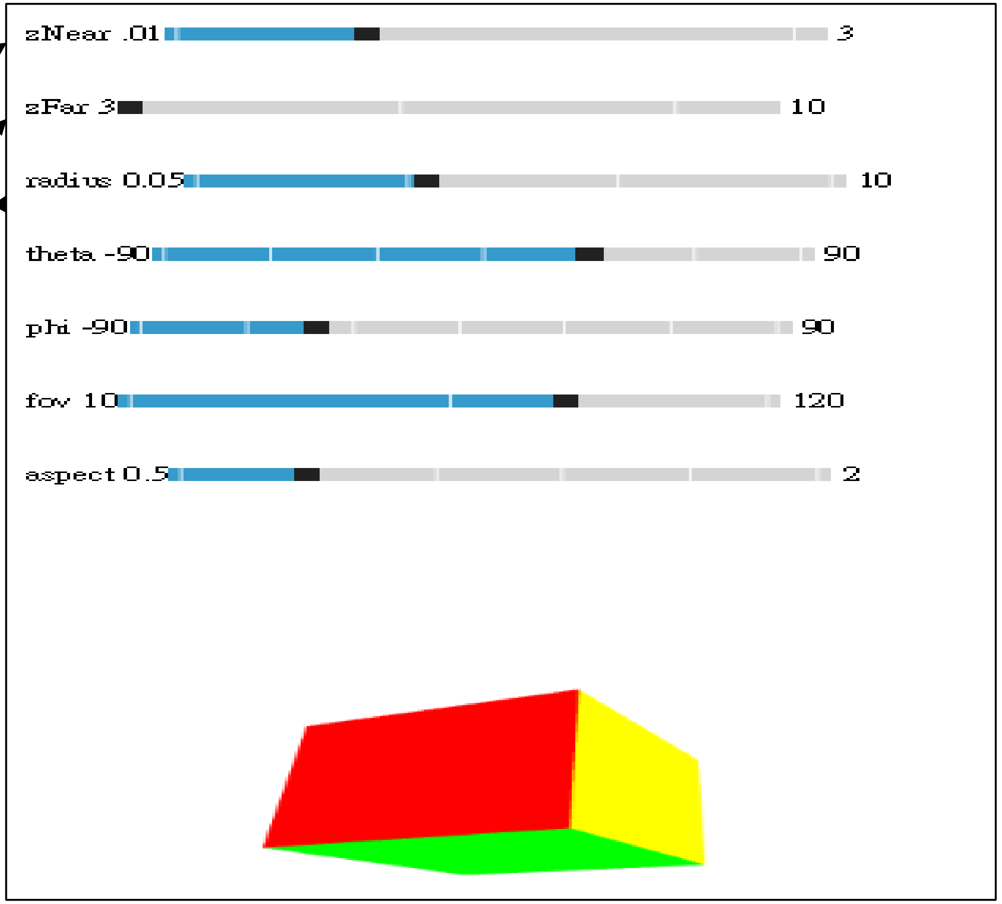
Sample Programs: perspective2.html, perspective2.js



Use slide bars instead of buttons

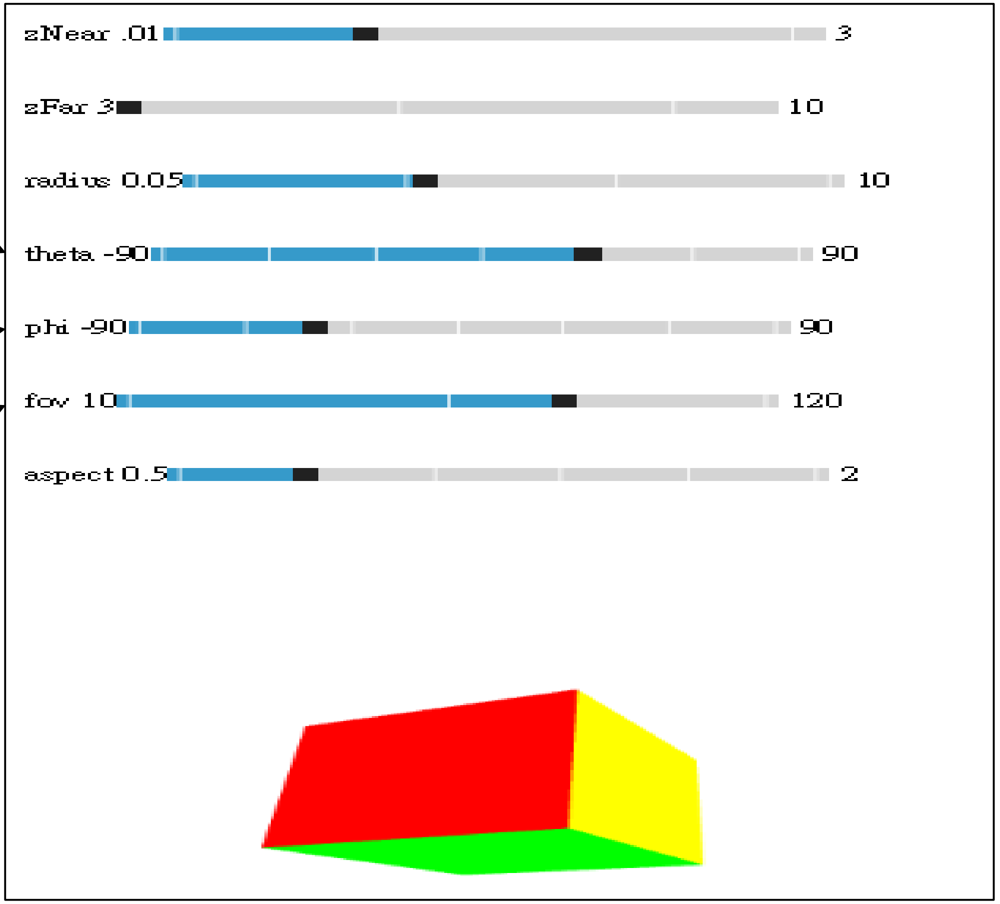
perspective2.html (1/6)

```
<!DOCTYPE html>
<html>
<p> </p>
<div>
zNear .01<input id="zNearSlider" type="range"
 min=".01" max="3" step="0.1" value="0.3" />
 3
</div>
<div>
zFar 3<input id="zFarSlider" type="range"
 min="3" max="10" step="3.0" value="3" />
 10
</div>
<div>
radius 0.05<input id="radiusSlider" type="range"
 min="0.05" max="10" step="0.1" value="4" />
 10
</div>
```



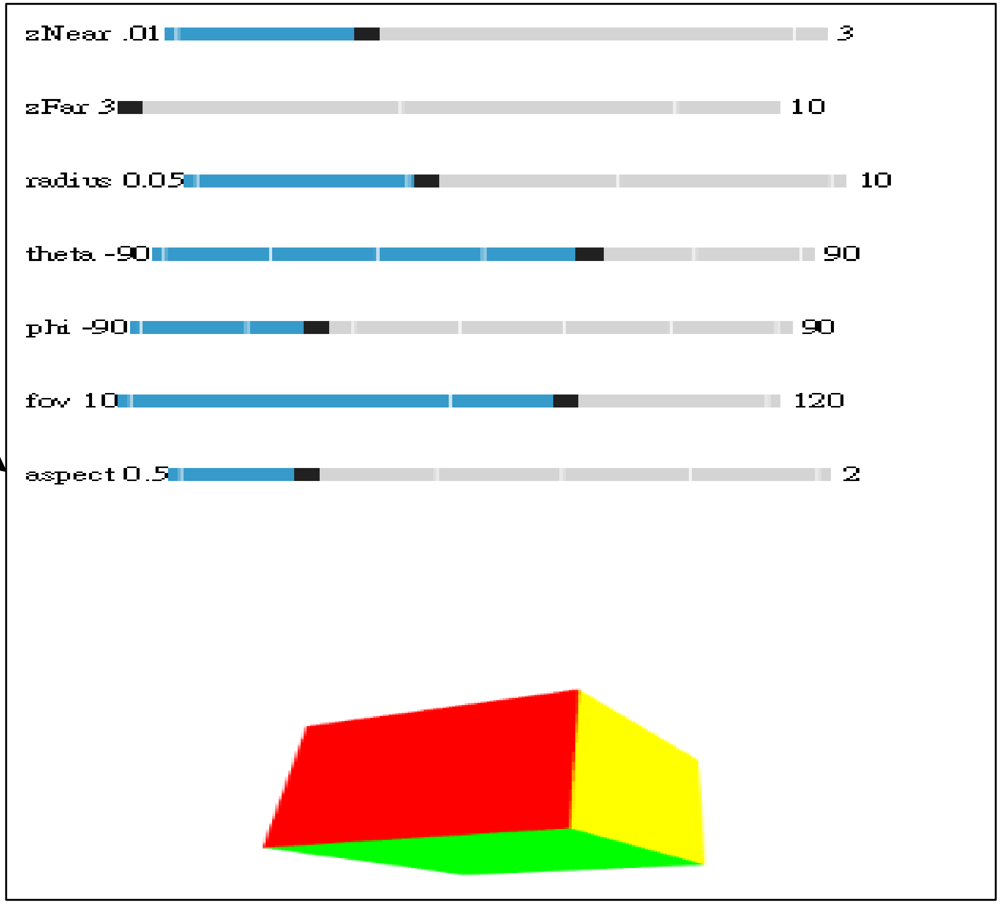
perspective2.html (2/6)

```
<div>
theta -90<input id="thetaSlider" type="range"
 min="-90" max="90" step="5" value="0" />
 90
</div>
<div>
phi -90<input id="phiSlider" type="range"
 min="-90" max="90" step="5" value="0" />
 90
</div>
<div>
fov 10<input id="fovSlider" type="range"
 min="10" max="120" step="5" value="45" />
 120
</div>
```



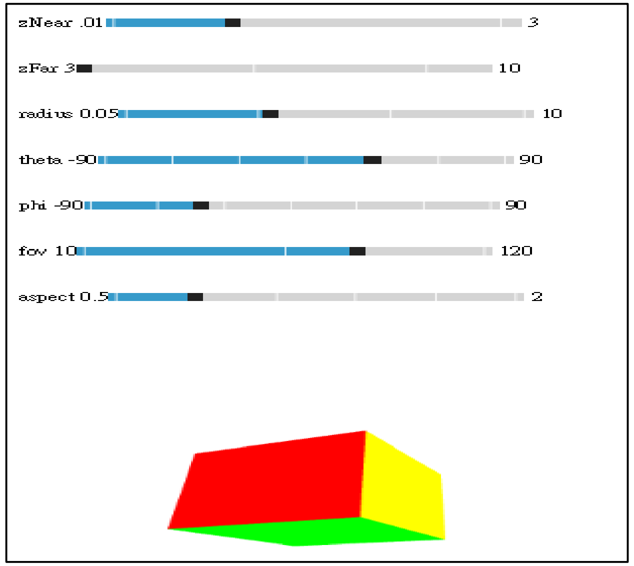
perspective2.html (3/6)

```
<div>
aspect 0.5<input id="aspectSlider" type="range"
 min="0.5" max="2" step="0.1" value="1" />
 2
</div>
```



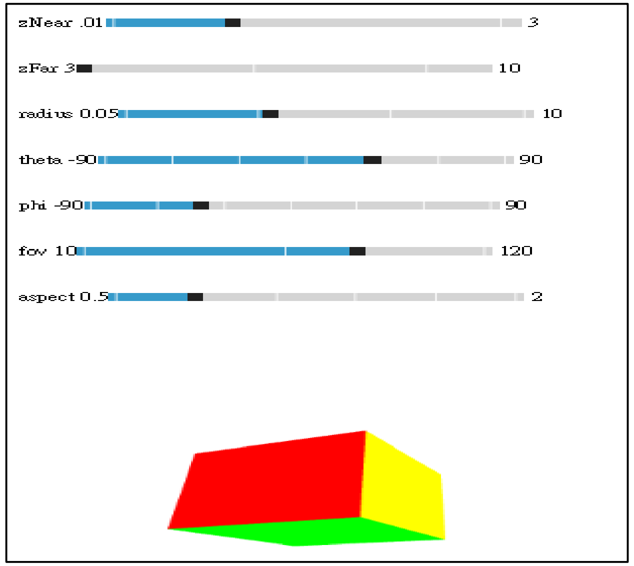
perspective2.html (4/6)

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

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

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

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



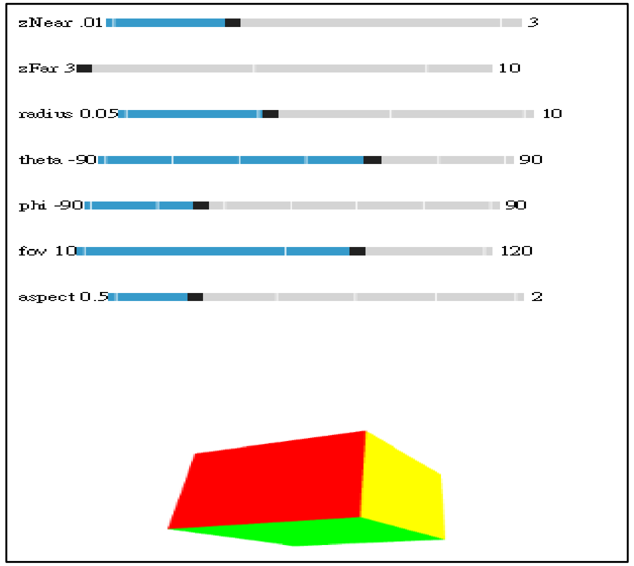
perspective2.html (5/6)

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

#ifdef GL_ES
precision highp float;
#endif

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="perspective2.js"></script>
```



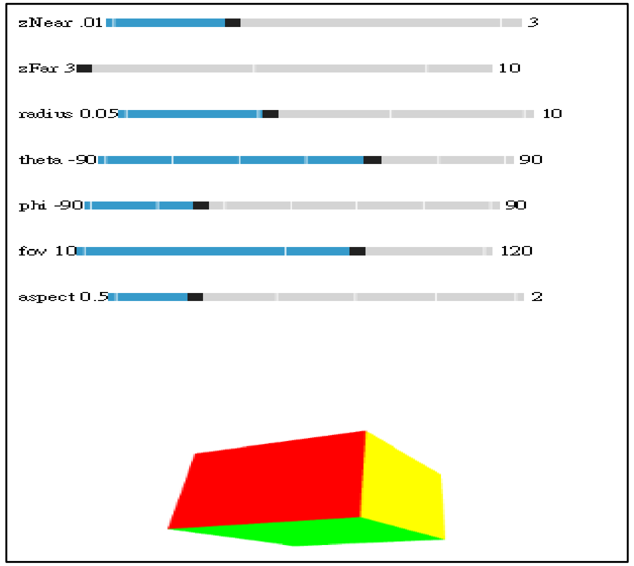
perspective2.html (6/6)

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

</body>
</html>
```



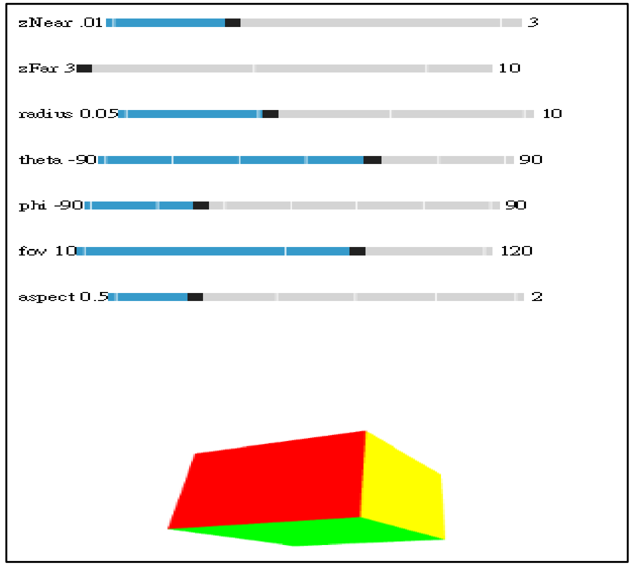
perspective2.js (1/11)

```
var canvas;
var gl;

var NumVertices  = 36;

var pointsArray = [];
var colorsArray = [];

var vertices = [
    vec4(-0.5, -0.5,  1.5, 1.0),
    vec4(-0.5,  0.5,  1.5, 1.0),
    vec4( 0.5,  0.5,  1.5, 1.0),
    vec4( 0.5, -0.5,  1.5, 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)
];
```



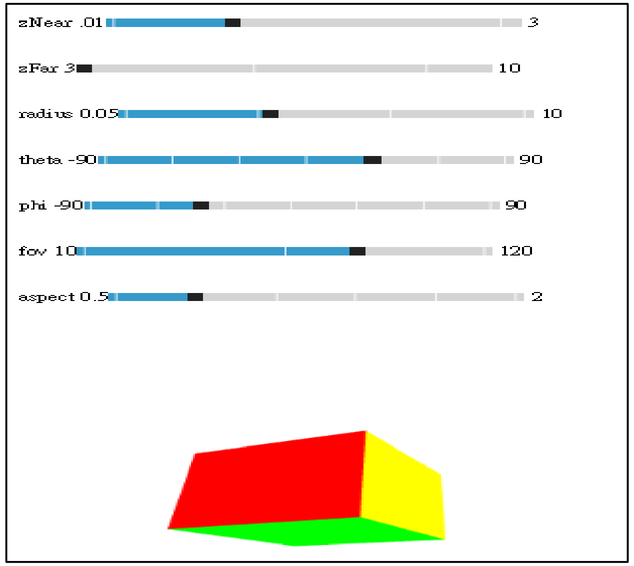
perspective2.js (2/11)

```
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
];
```



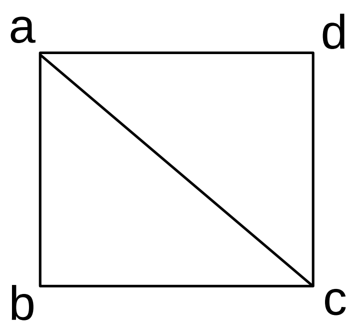
perspective2.js (3/11)

```
var near = 0.3;
var far = 3.0;
var radius = 4.0;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;

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

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var eye;
const at  = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);
```

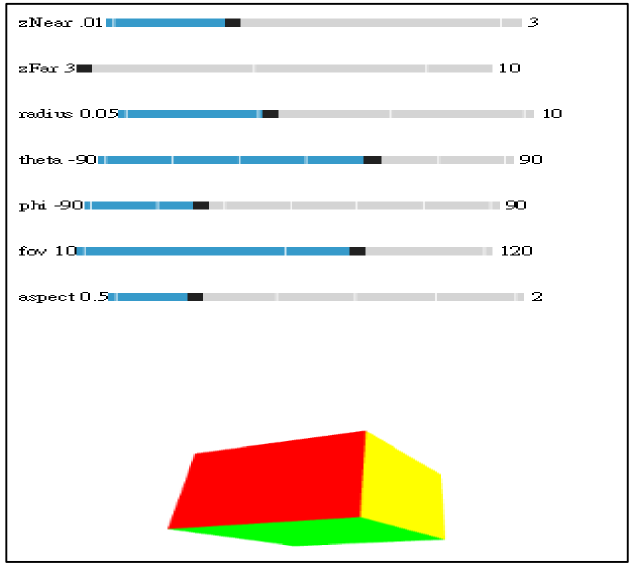


perspective2.js (4/11)

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

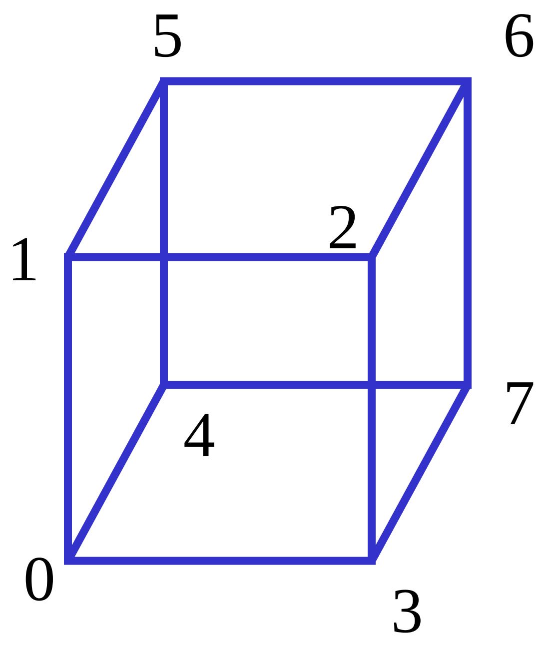


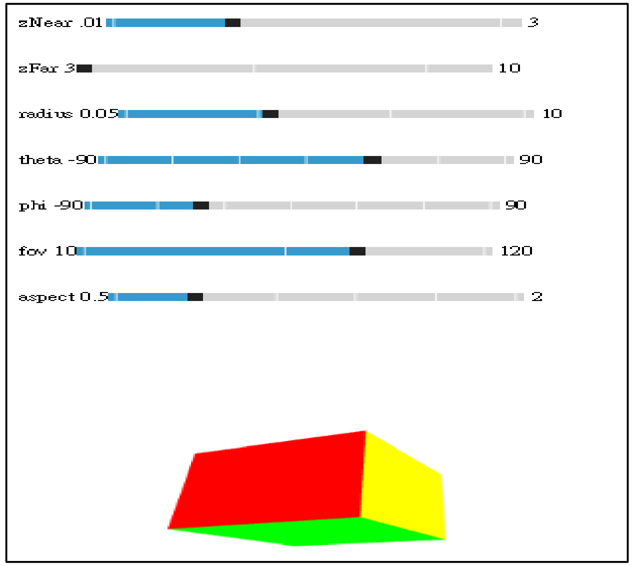
gl.TRIANGLES



perspective2.js (5/11)

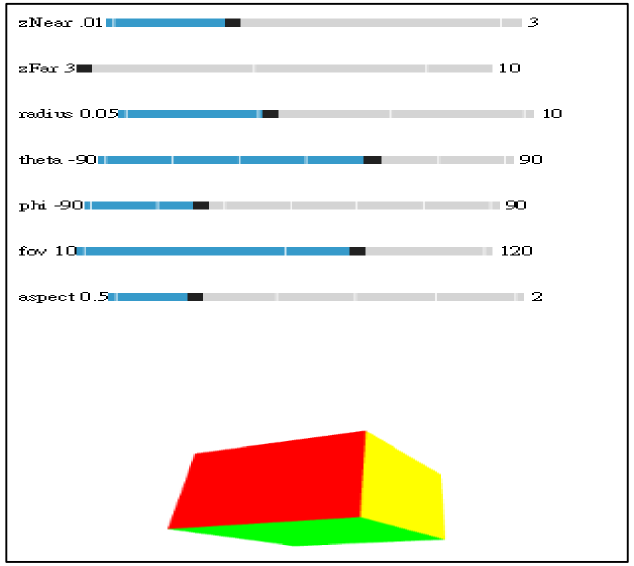
```
function colorCube()
{
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```





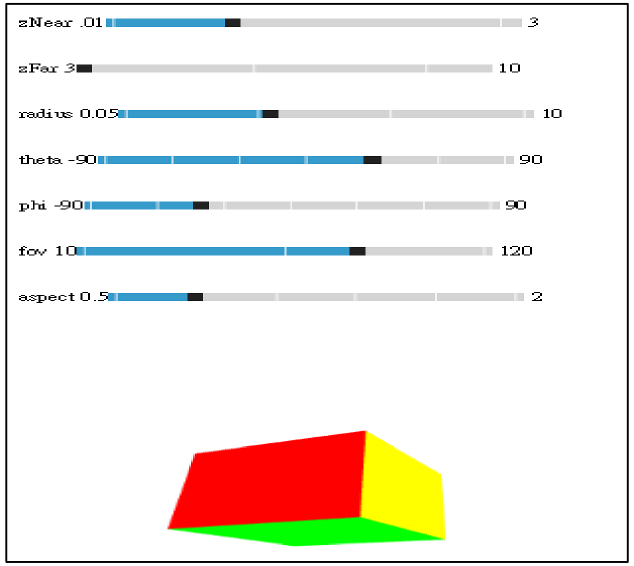
perspective2.js (6/11)

```
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 );

    aspect =  canvas.width/canvas.height;

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

    gl.enable(gl.DEPTH_TEST);
```



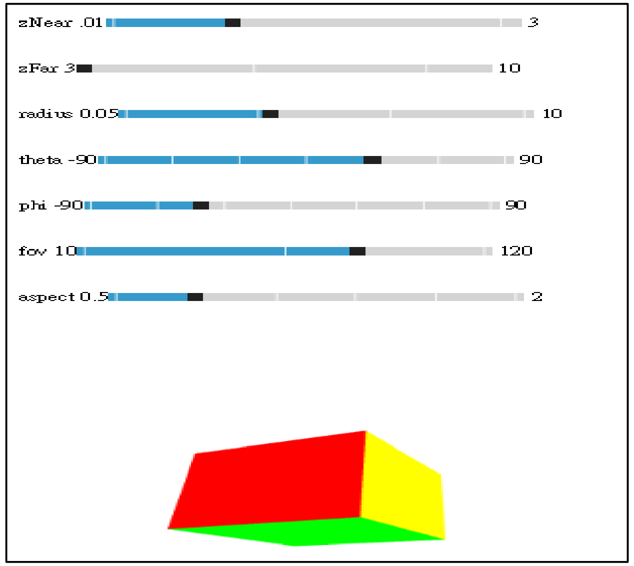
perspective2.js (7/11)

```
//
//  Load shaders and initialize attribute buffers
//
var 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);
```



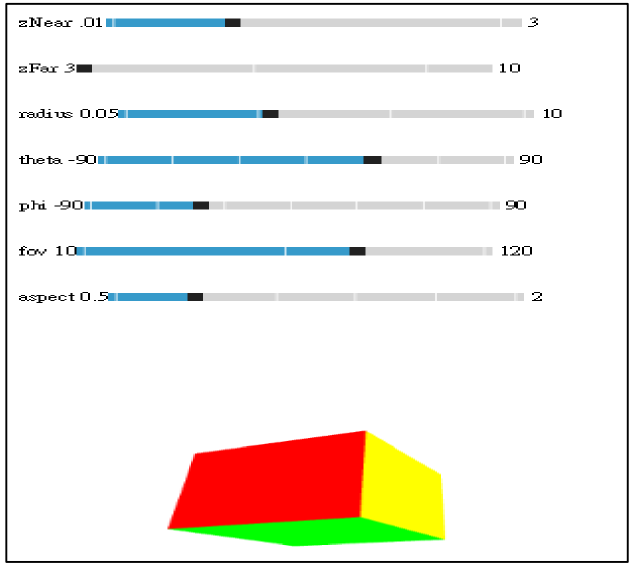
perspective2.js (8/11)

```
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 );

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



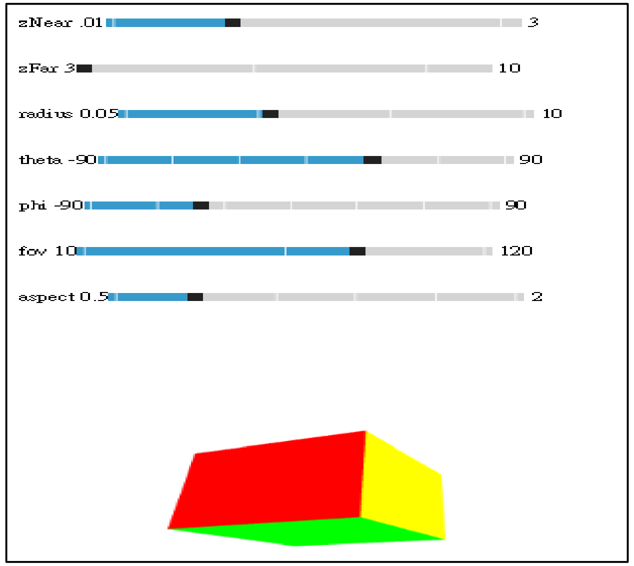
perspective2.js (9/11)

```
// sliders for viewing parameters

    document.getElementById("zFarSlider").onchange = function() {
        far = event.srcElement.value;
    };
    document.getElementById("zNearSlider").onchange = function() {
        near = event.srcElement.value;
    };
    document.getElementById("radiusSlider").onchange = function() {
       radius = event.srcElement.value;
    };
    document.getElementById("thetaSlider").onchange = function() {
        theta = event.srcElement.value* Math.PI/180.0;
    };
    document.getElementById("phiSlider").onchange = function() {
        phi = event.srcElement.value* Math.PI/180.0;
    };
```



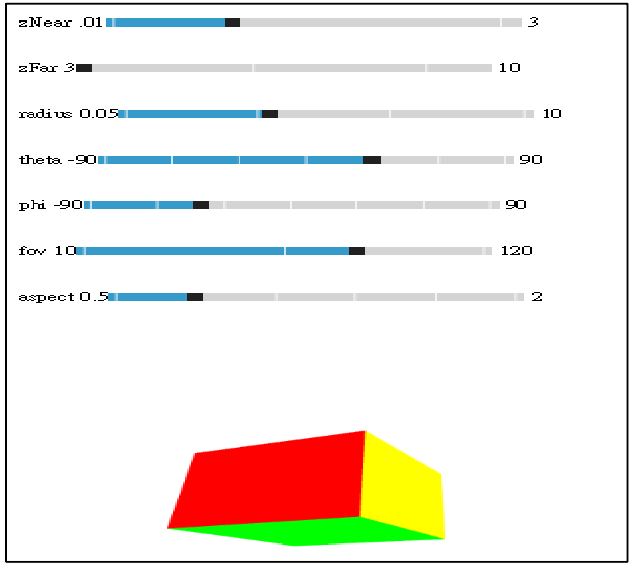
perspective2.js (10/11)

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

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

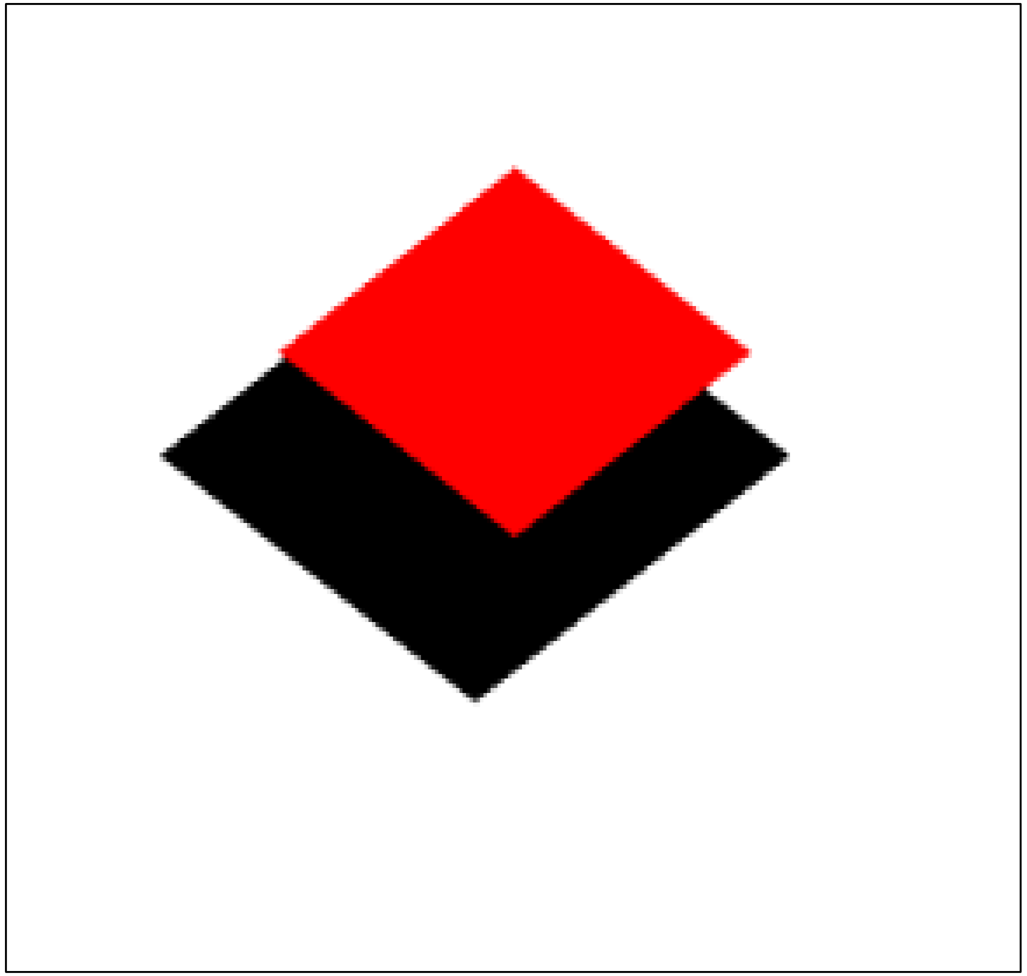


perspective2.js (11/11)

```
var render = function() {

    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
                       radius*Math.sin(theta)*Math.sin(phi),
                       radius*Math.cos(theta));
    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()
```



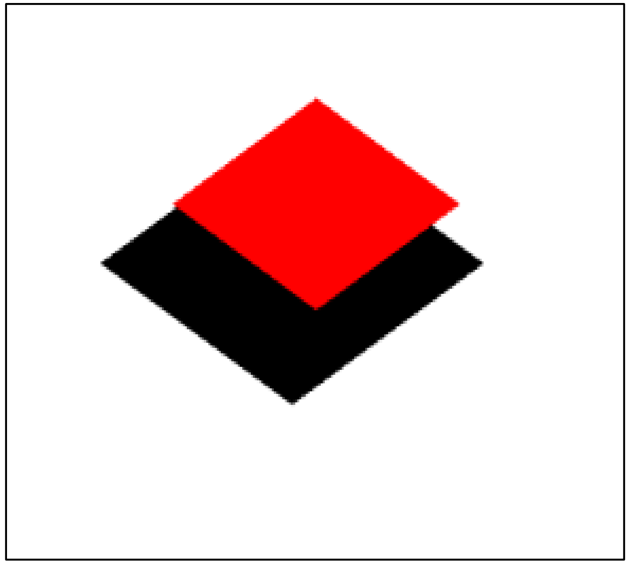
Sample Programs: shadow.html, shadow.js



projective shadow of a square onto y = 0 plane with moving light

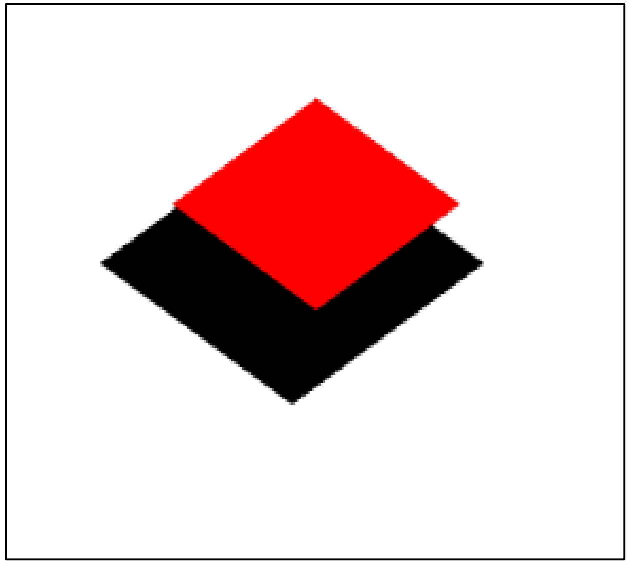
shadow.html (1/3)

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

attribute  vec4 vPosition;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

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

}
</script>
```



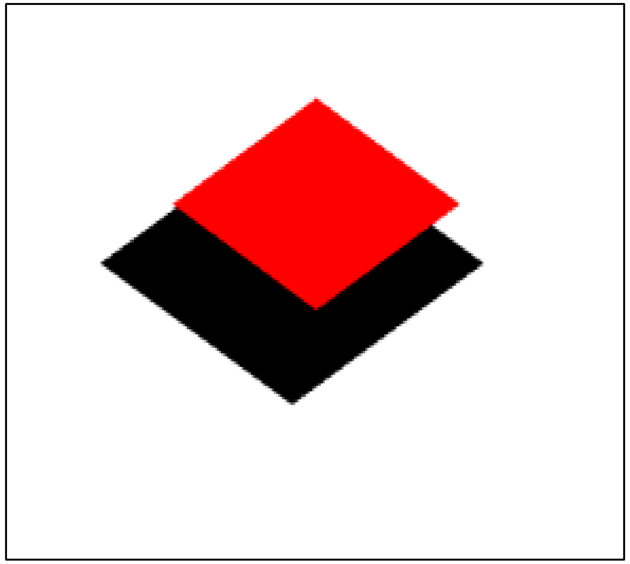
shadow.html (2/3)

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

precision mediump float;

uniform 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="shadow.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
```



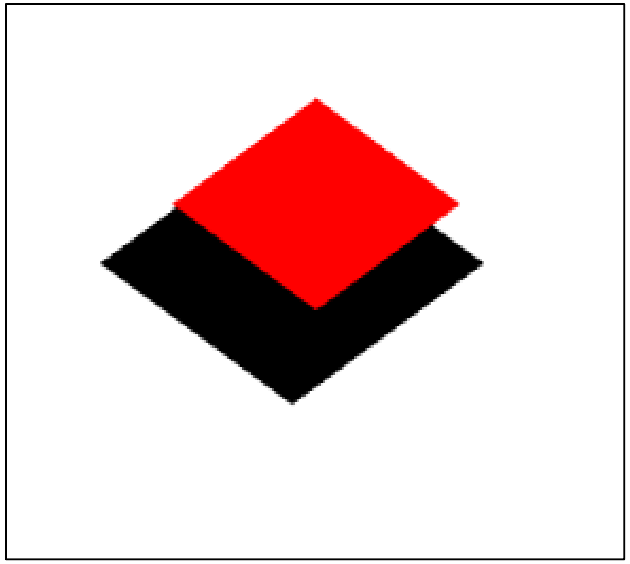
shadow.html (3/3)

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

</body>
</html>
```



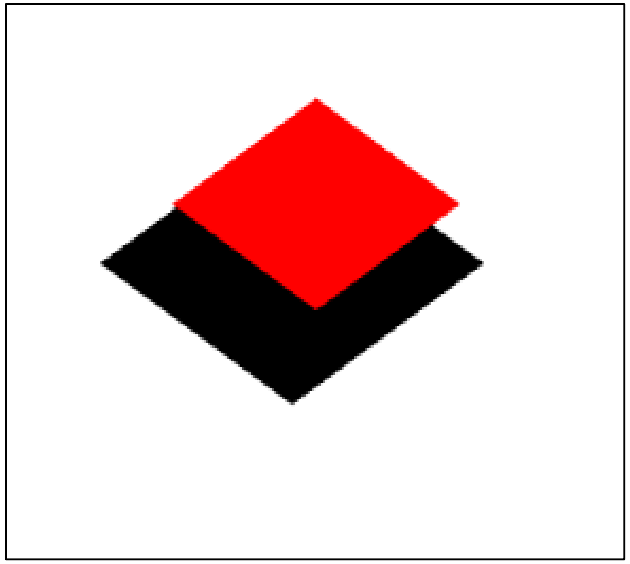
shadow.js (1/9)

```
var canvas;
var gl;

var pointsArray = [];

var near = -4;
var far = 4;

var theta  = 0.0;

var left = -2.0;
var right = 2.0;
var ytop = 2.0;
var bottom = -2.0;
```



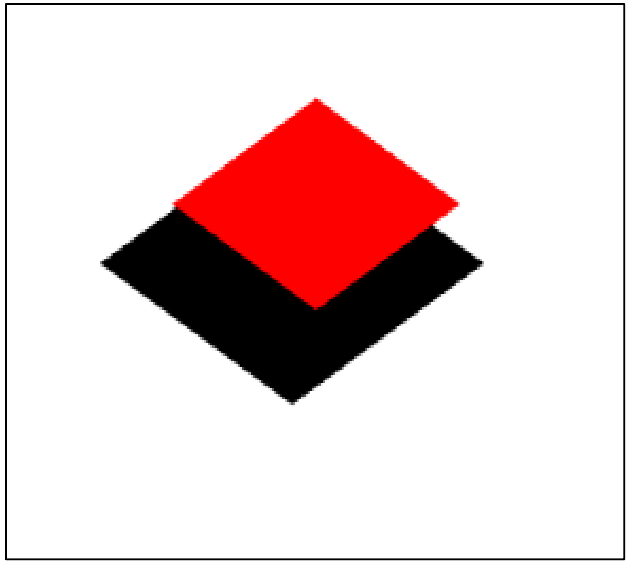
shadow.js (2/9)

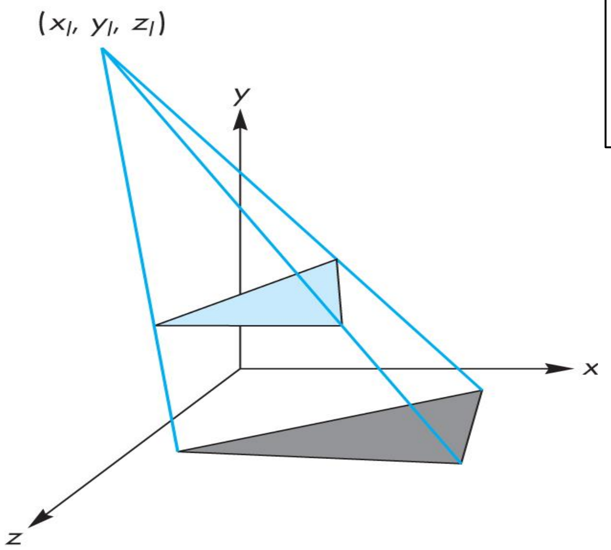
```
var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;

var fColor;

var eye, at, up;
var light;

var m;

var red;
var black;
```



shadow.js (3/9)

```
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);

    light = vec3(0.0, 2.0, 0.0);
```



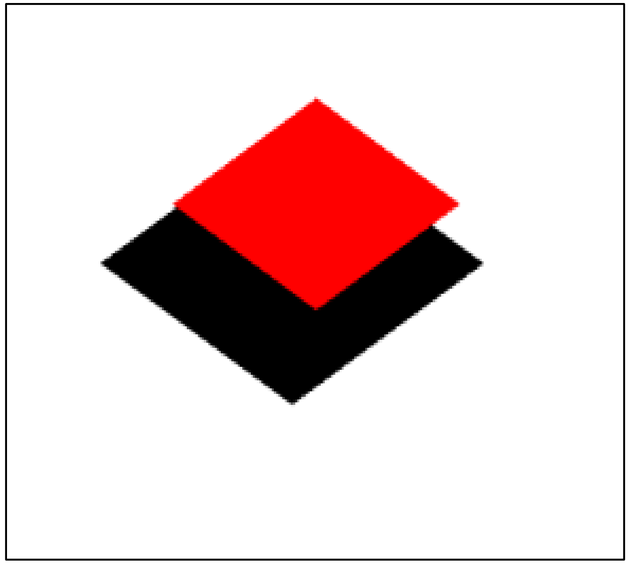


shadow.js (4/9)

```
// matrix for shadow projection

m = mat4();
m[3][3] = 0;
m[3][1] = -1/light[1];


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

// color square red and shadow black

red = vec4(1.0, 0.0, 0.0, 1.0);
black = vec4(0.0, 0.0, 0.0, 1.0);
```

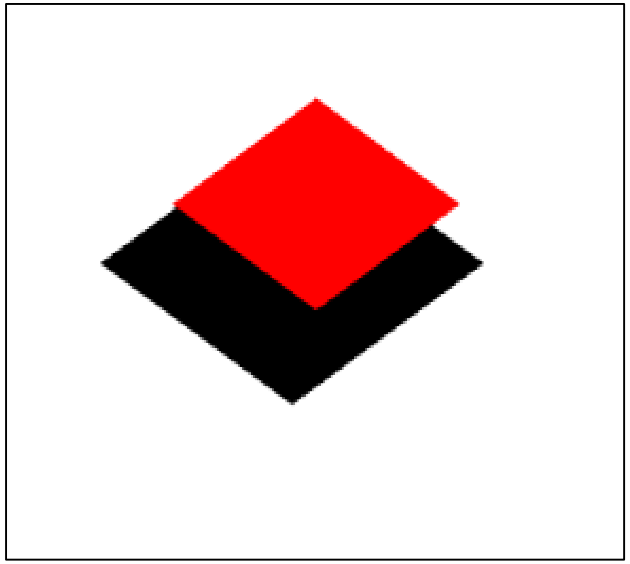
$$m = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{-y_l} & 0 & 0 \end{bmatrix}$$

Light source $L(x_l, y_l, z_l, 1)$



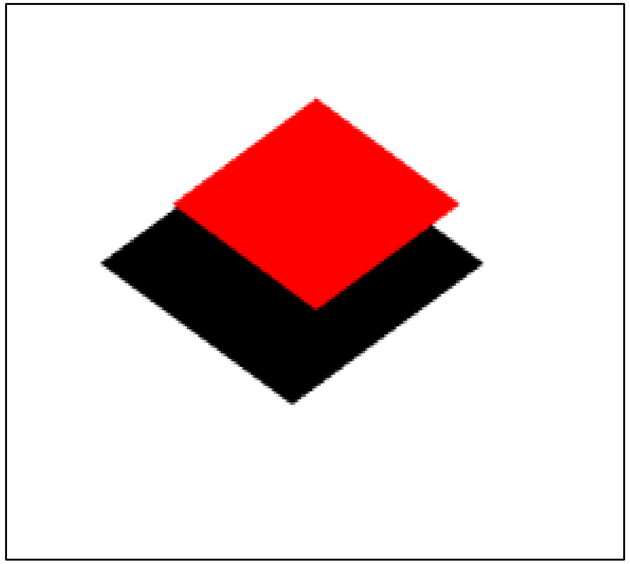
shadow.js (5/9)

```
// square
pointsArray.push(vec4( -0.5,  0.5,  -0.5, 1.0 ));
pointsArray.push(vec4( -0.5,  0.5,   0.5, 1.0 ));
pointsArray.push(vec4(  0.5,  0.5,   0.5, 1.0 ));
pointsArray.push(vec4(  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 );

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 );
```



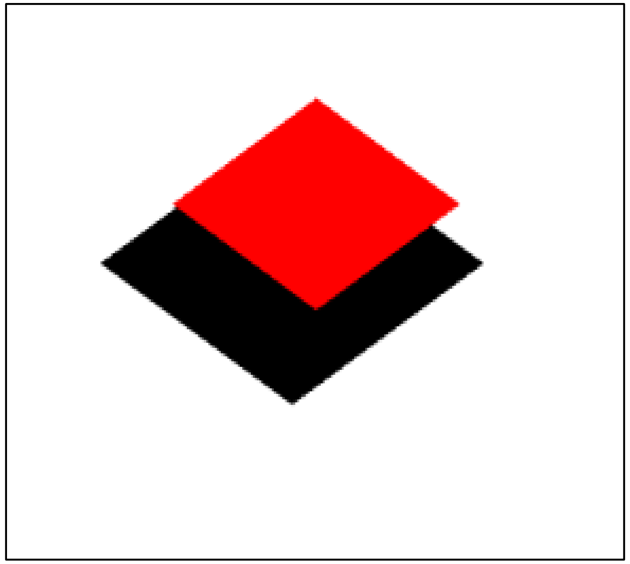
shadow.js (6/9)

```
fColor = gl.getUniformLocation(program, "fColor");

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

projectionMatrix = ortho(left, right, bottom, ytop, near, far);
gl.uniformMatrix4fv( projectionMatrixLoc, false, flatten(projectionMatrix) );

render();

}   // end of window.onload
```



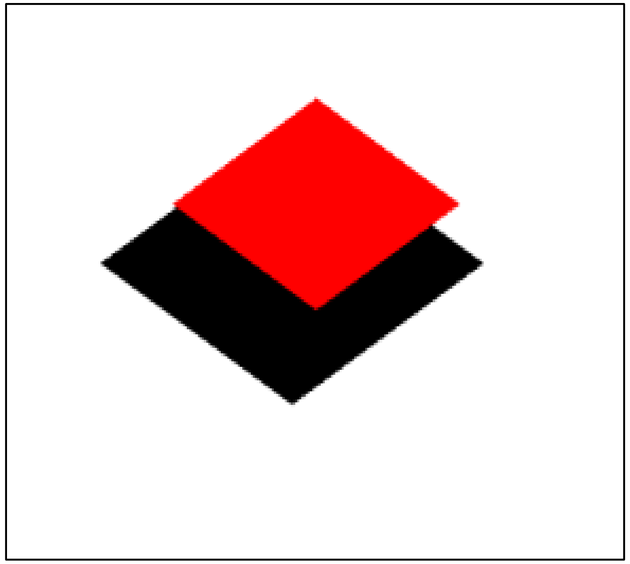
shadow.js (7/9)

```
var render = function() {

    theta += 0.1;
    if(theta > 2*Math.PI) theta -= 2*Math.PI;

    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    // model-view matrix for square

    modelViewMatrix = lookAt(eye, at, up);

    // send color and matrix for square then render

    gl.uniformMatrix4fv( modelViewMatrixLoc, false, flatten(modelViewMatrix) );
    gl.uniform4fv(fColor, flatten(red));
    gl.drawArrays(gl.TRIANGLE_FAN, 0, 4);
```



shadow.js (8/9)

```
// rotate light source

light[0] = Math.sin(theta);
light[2] = Math.cos(theta);

// model-view matrix for shadow then render

modelViewMatrix = mult(modelViewMatrix, translate(light[0], light[1], light[2]));
modelViewMatrix = mult(modelViewMatrix, m);
modelViewMatrix = mult(modelViewMatrix, translate(-light[0], -light[1], -light[2]));
```

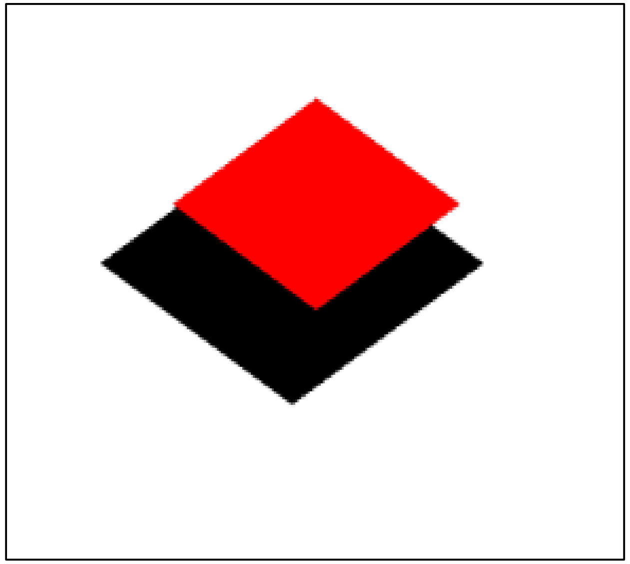


$$\boxed{\begin{array}{l} \text{modelViewMatrix} = T(x_l, y_l, z_l) \cdot \text{m} \cdot T(-x_l, -y_l, -z_l) \\ P''^T = \text{modelViewMatrix} \cdot P^T \end{array}}$$

$$\text{m} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{-y_l} & 0 & 0 \end{bmatrix}$$

shadow.js (9/9)

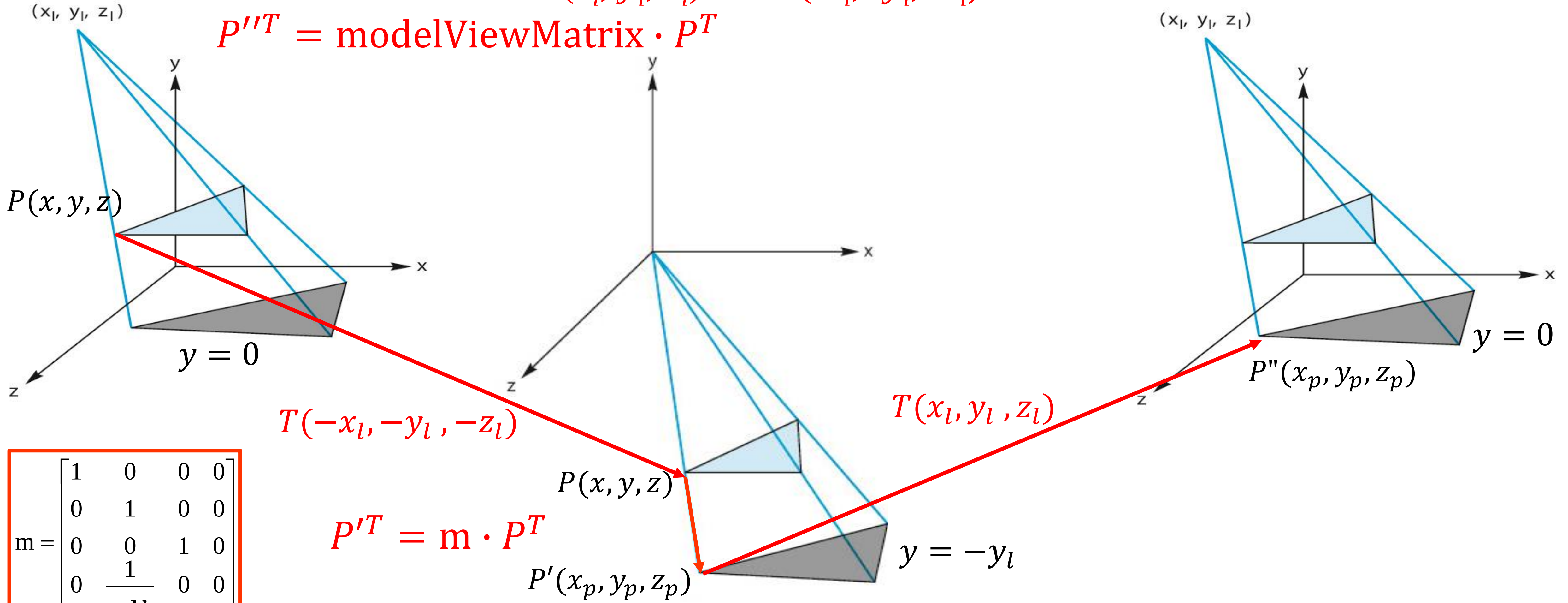
```
// send color and matrix for shadow

gl.uniformMatrix4fv( modelViewMatrixLoc, false, flatten(modelViewMatrix) );
gl.uniform4fv(fColor, flatten(black));
gl.drawArrays(gl.TRIANGLE_FAN, 0, 4);

requestAnimFrame(render);

}   // end of render()
```



A note on shadow.js

modelViewMatrix=$T(x_l, y_l, z_l) \cdot \mathrm{m} \cdot T(-x_l, -y_l, -z_l)$

$P''^T = \text{modelViewMatrix} \cdot P^T$



$P'^T = \mathrm{m} \cdot P^T$

$$\mathrm{m} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{-y_l} & 0 & 0 \end{bmatrix}$$