



Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research,
Technology and Science Laboratory

University of New Mexico



The University of New Mexico

Orthogonal Projection Matrices

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico



The University of New Mexico

Objectives

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

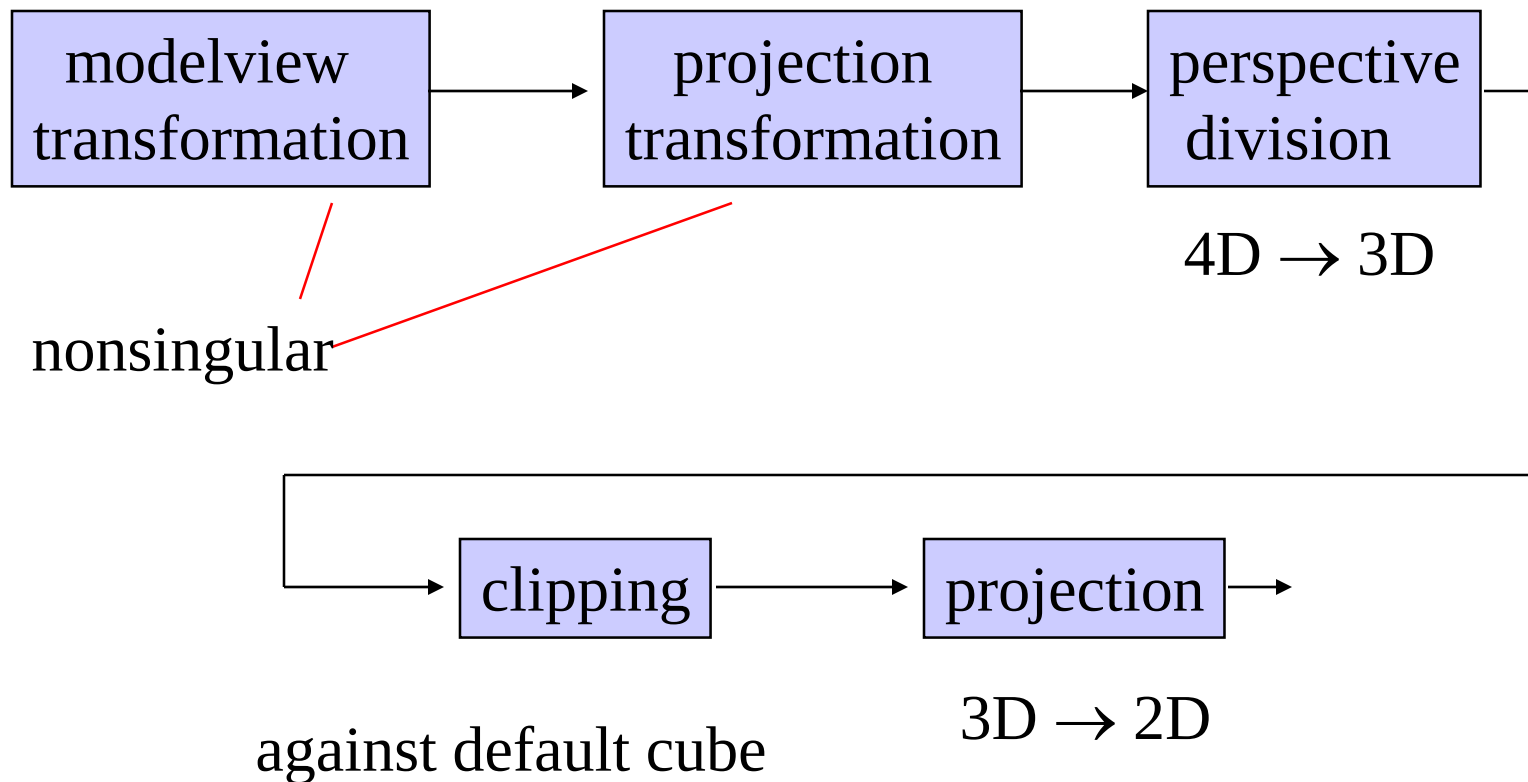


Normalization

- 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

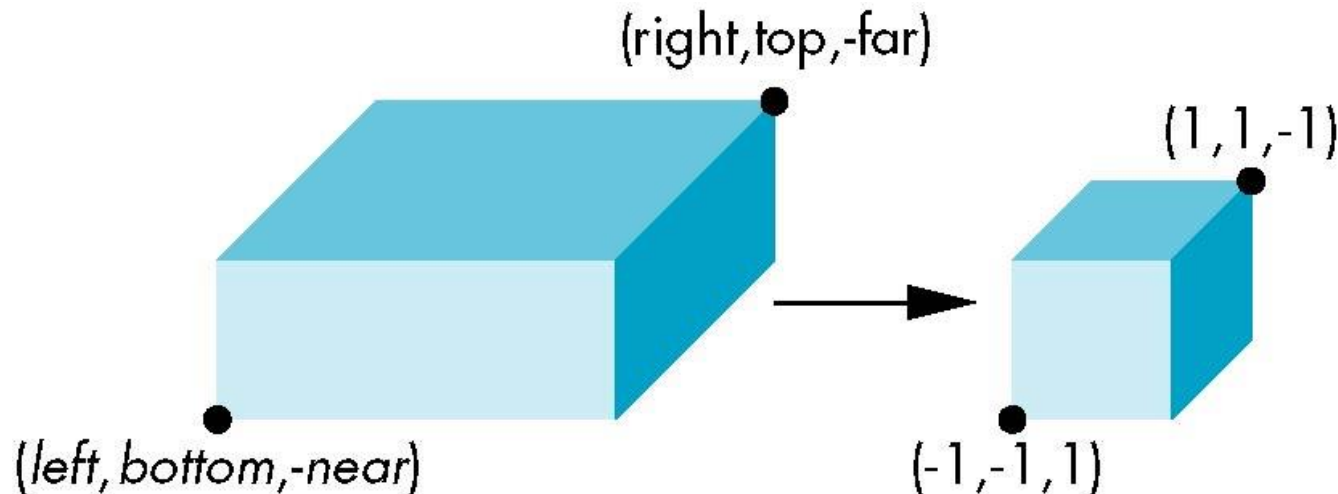


The University of New Mexico

Orthogonal Normalization

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

normalization $\Rightarrow$ 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

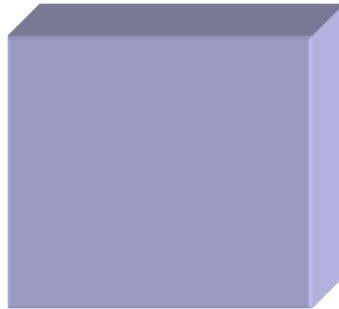
$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Hence, general orthogonal projection in 4D is

$$\mathbf{P} = \mathbf{M}_{\text{orth}} \mathbf{S} \mathbf{T}$$

Oblique Projections

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

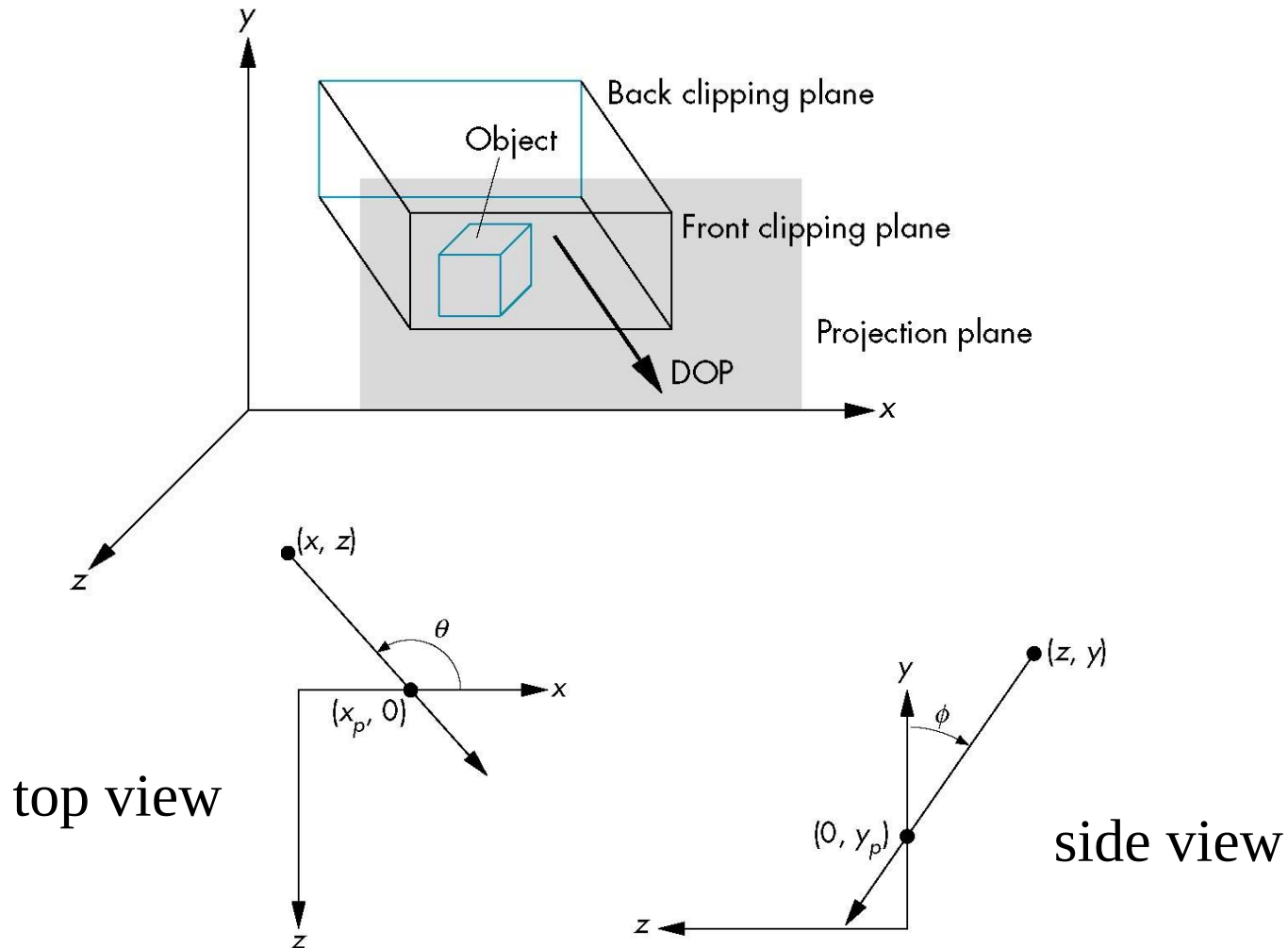


- However if we look at the example of the cube it appears that the cube has been sheared
- Oblique Projection = Shear + Orthogonal Projection



The University of New Mexico

General Shear





Shear Matrix

xy shear (z values unchanged)

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

Projection matrix

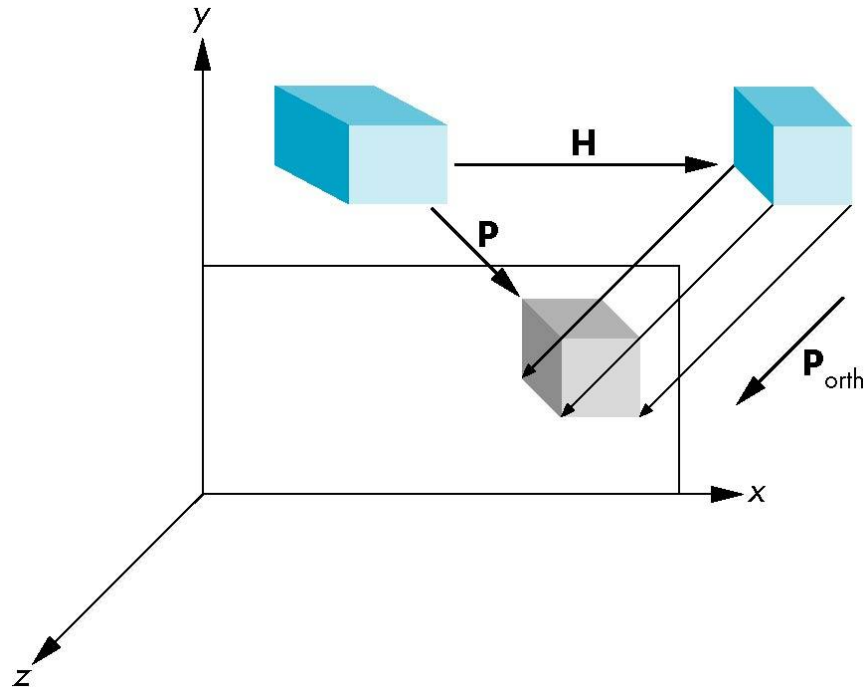
$$\mathbf{P} = \mathbf{M}_{\text{orth}} \mathbf{H}(\theta, \phi)$$

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



The University of New Mexico

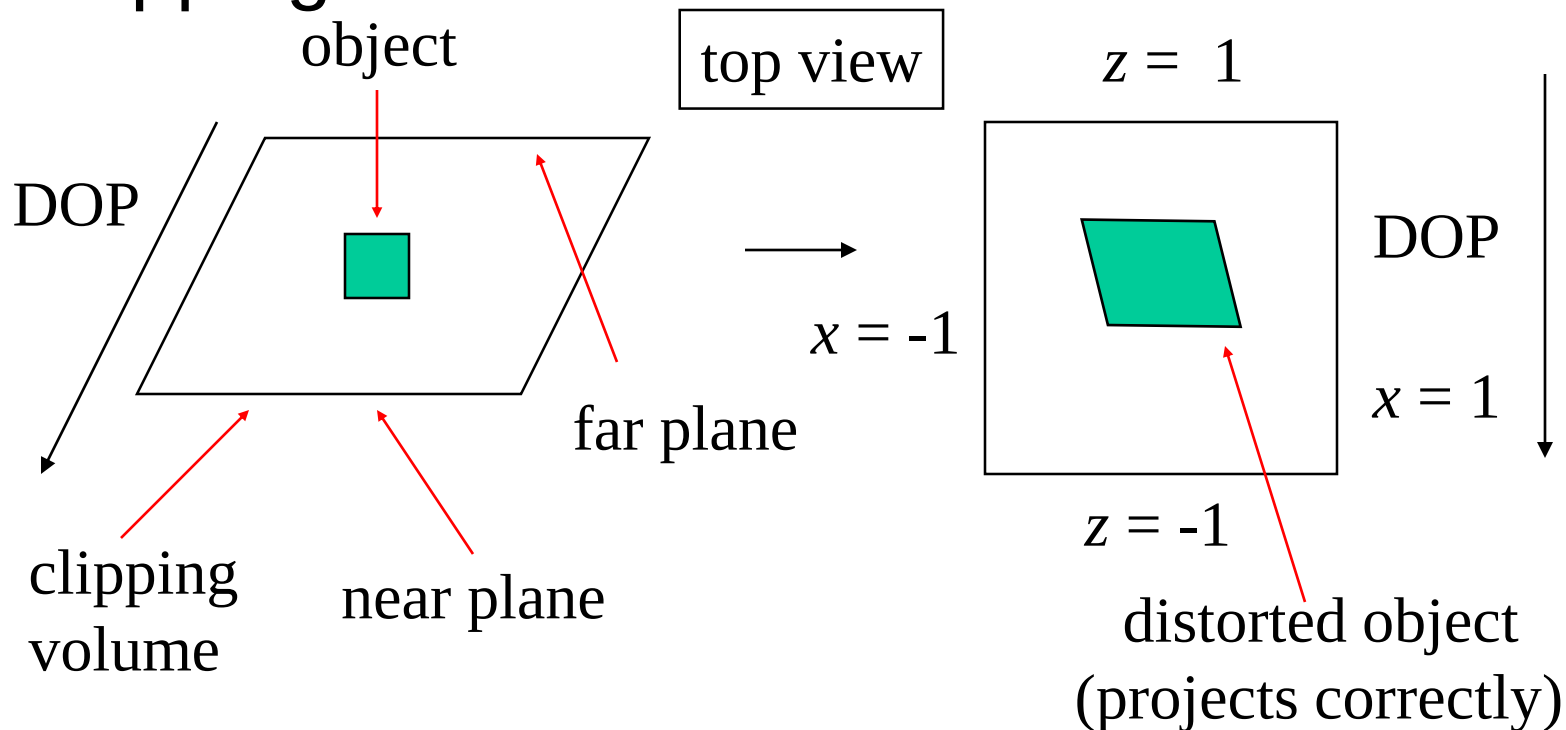
Equivalency





Effect on Clipping

- The projection matrix $\mathbf{P} = \mathbf{S}\mathbf{T}\mathbf{H}$ transforms the original clipping volume to the default clipping volume





Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research,
Technology and Science Laboratory

University of New Mexico



The University of New Mexico

Perspective Projection Matrices

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico



The University of New Mexico

Objectives

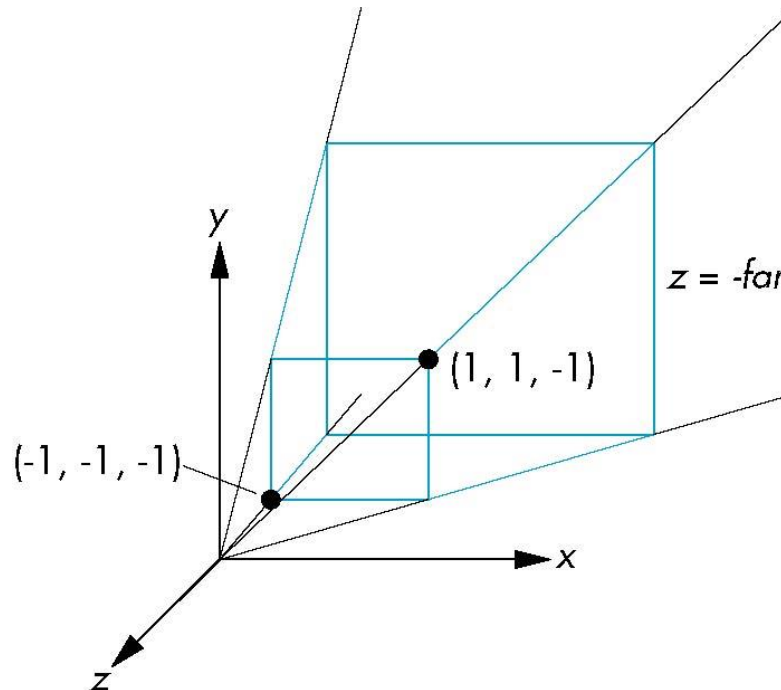
- Derive the perspective projection matrices used for standard WebGL projections



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

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

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

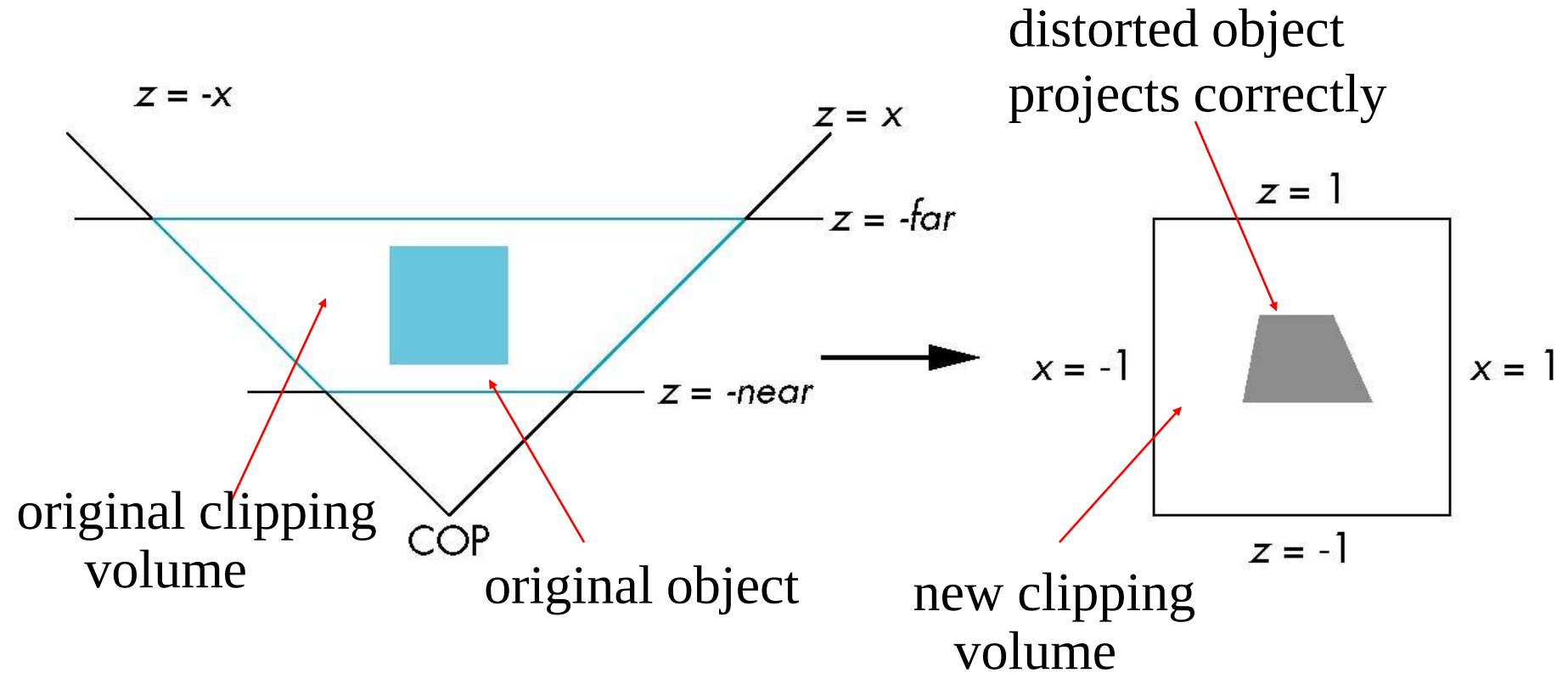
the near plane is mapped to $z = -1$

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





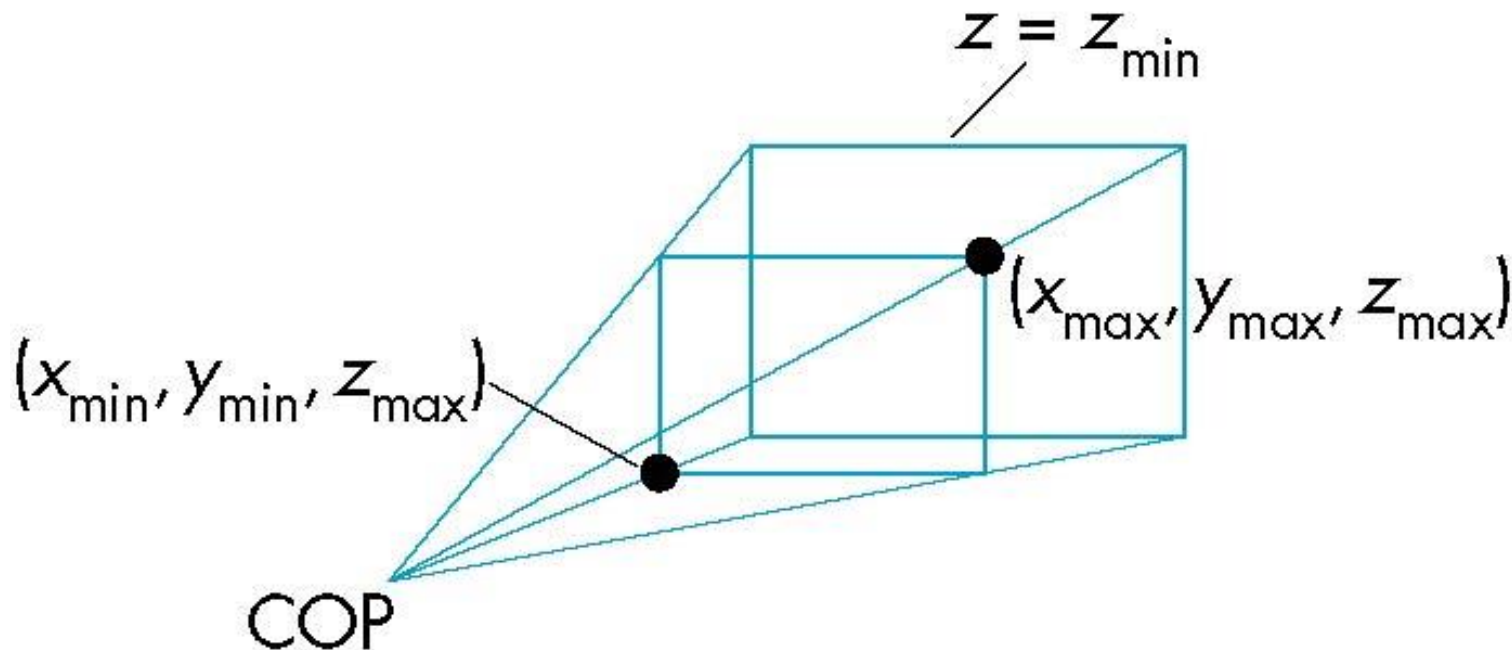
Normalization and Hidden-Surface Removal

- 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

$$\mathbf{P} = \mathbf{N}\mathbf{S}\mathbf{H}$$

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

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

perspective

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



Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research,
Technology and Science Laboratory

University of New Mexico



The University of New Mexico

Meshes

Ed Angel

Professor Emeritus of Computer Science
University of New Mexico



The University of New Mexico

Objective

Introduce techniques for displaying
polygonal meshes



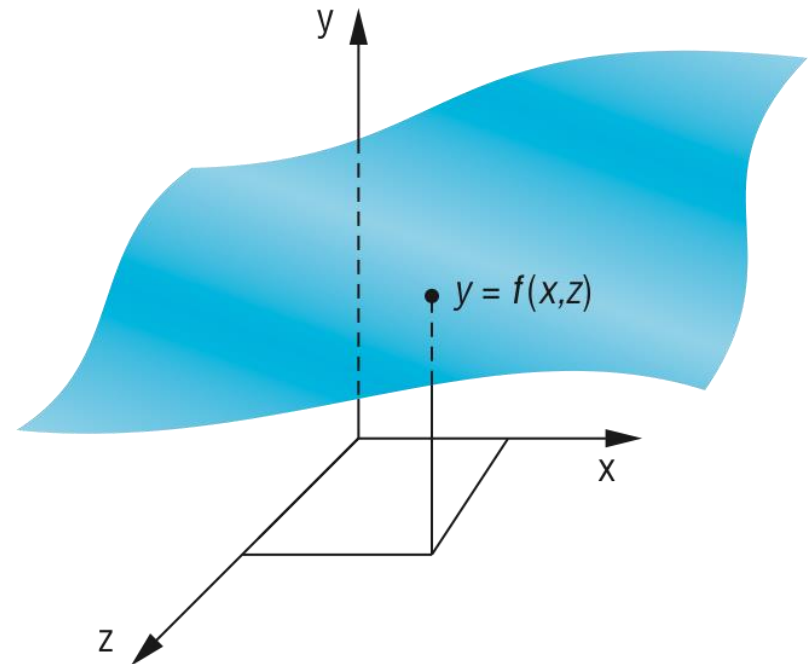
Meshes

- 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

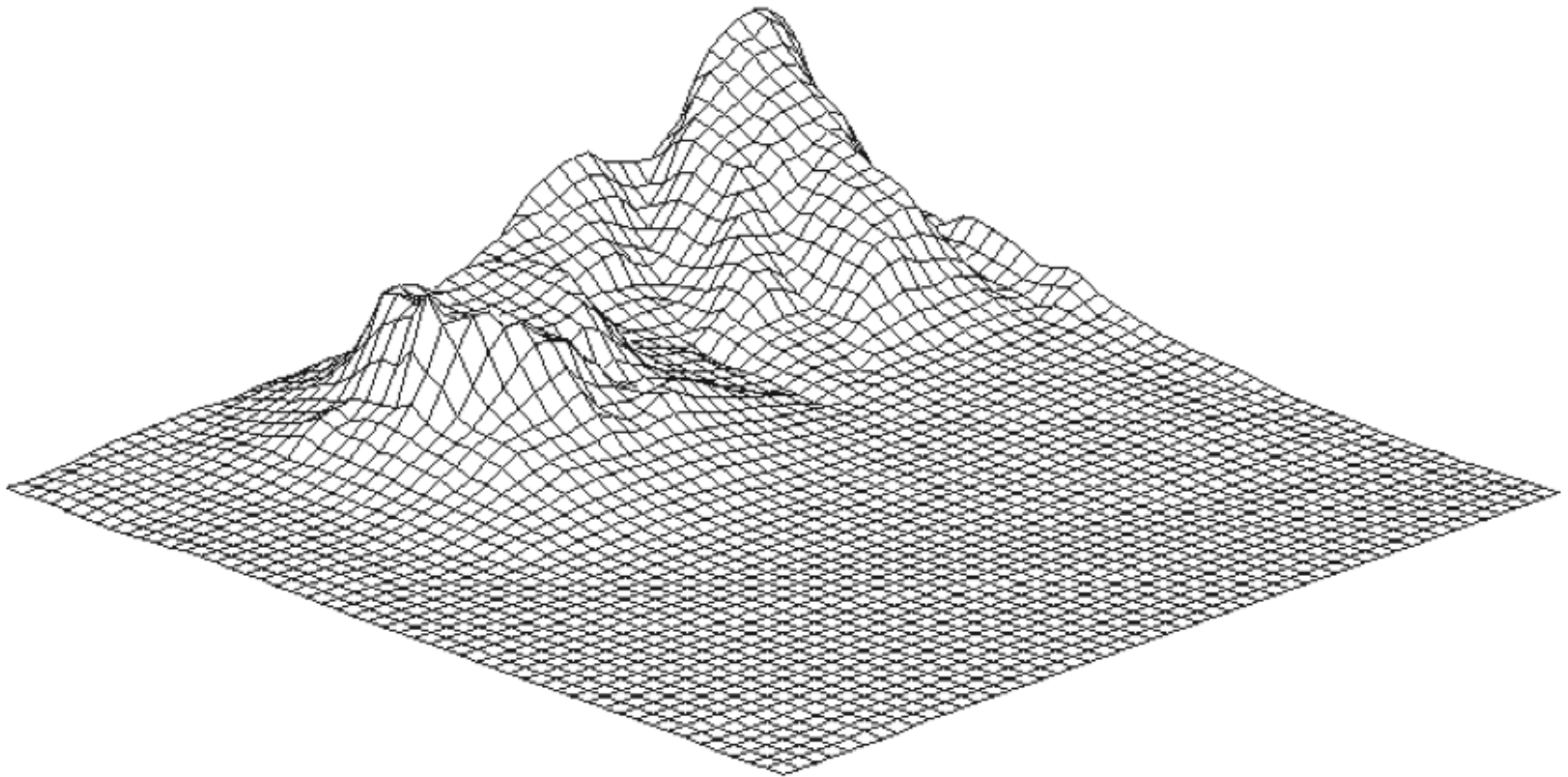
- For each (x, z) there is a unique y
- Sampling leads to an array of y values
- Display as quadrilateral or triangular mesh using strips





The University of New Mexico

Honolulu Plot Using Line Strips





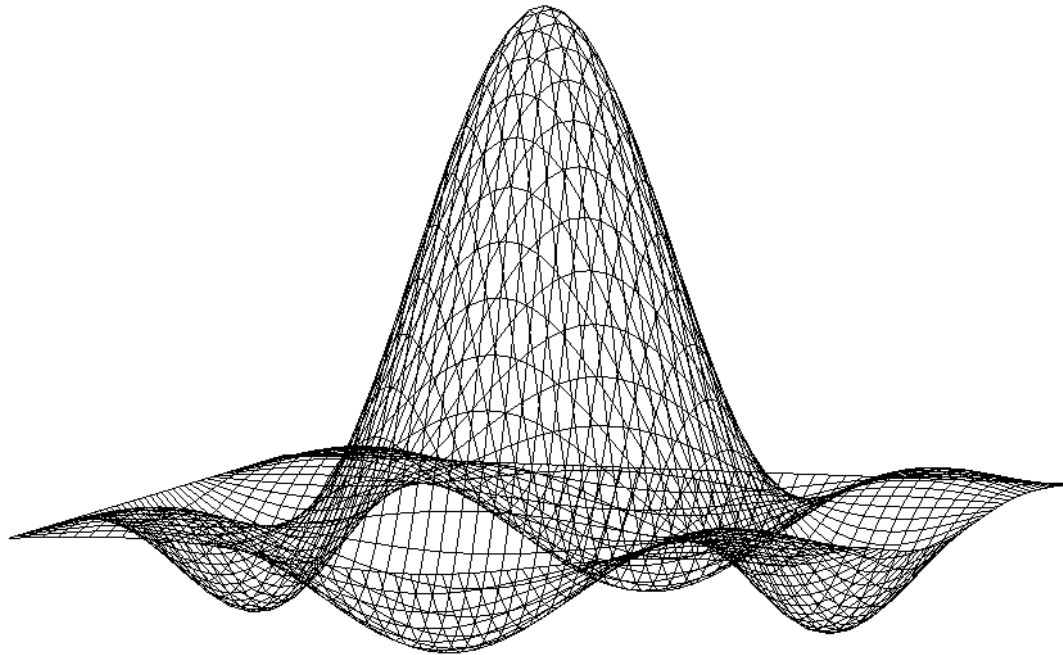
Plot 3D

- 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



The University of New Mexico

Lines on Back and Hidden Faces



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



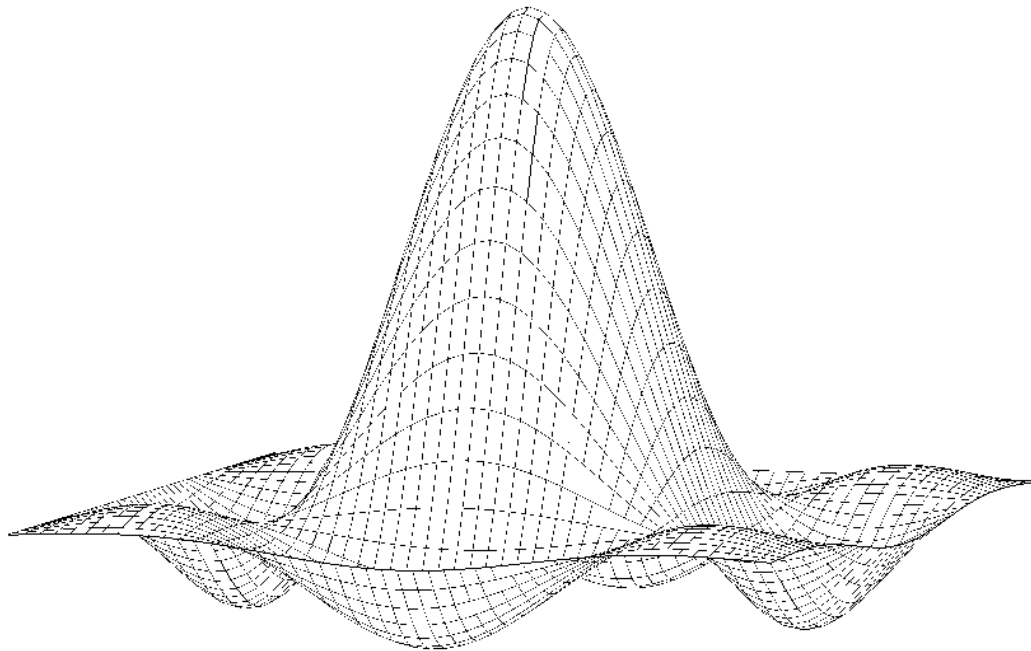
Using Polygons

- 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



The University of New Mexico

Hat Using Triangles and Lines





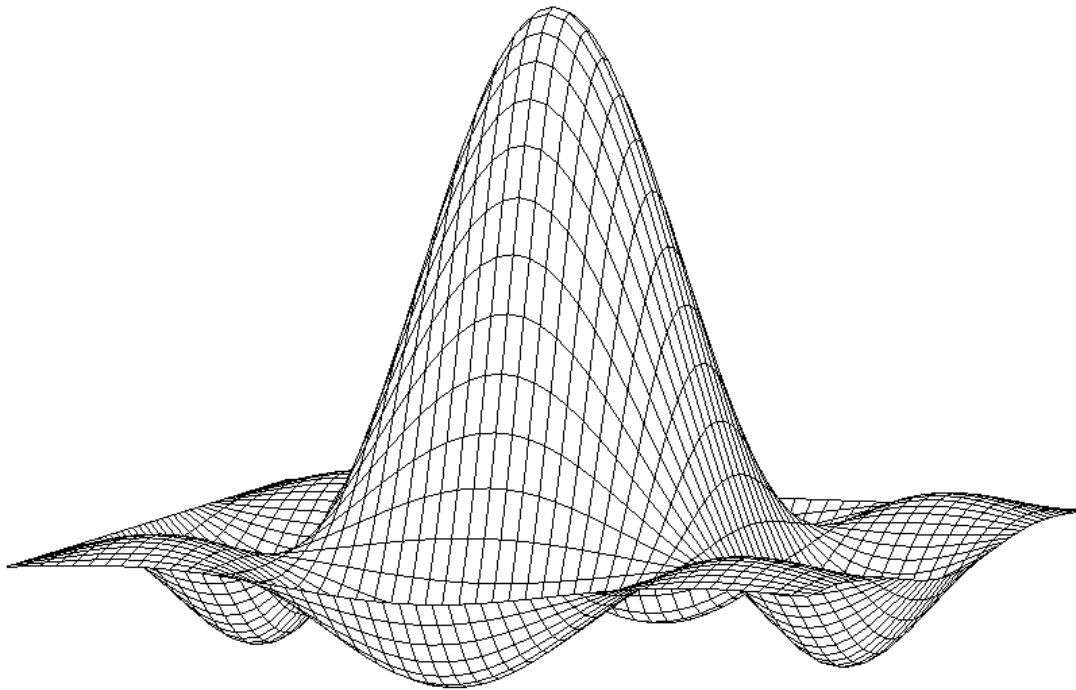
Polygon Offset

- 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



The University of New Mexico

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



Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research,
Technology and Science Laboratory

University of New Mexico



The University of New Mexico

Shadows

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico



The University of New Mexico

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 on an object at a time so shadows are not automatic
 - can use some tricks: projective shadows
 - multi-rendering: shadow maps and shadow volumes



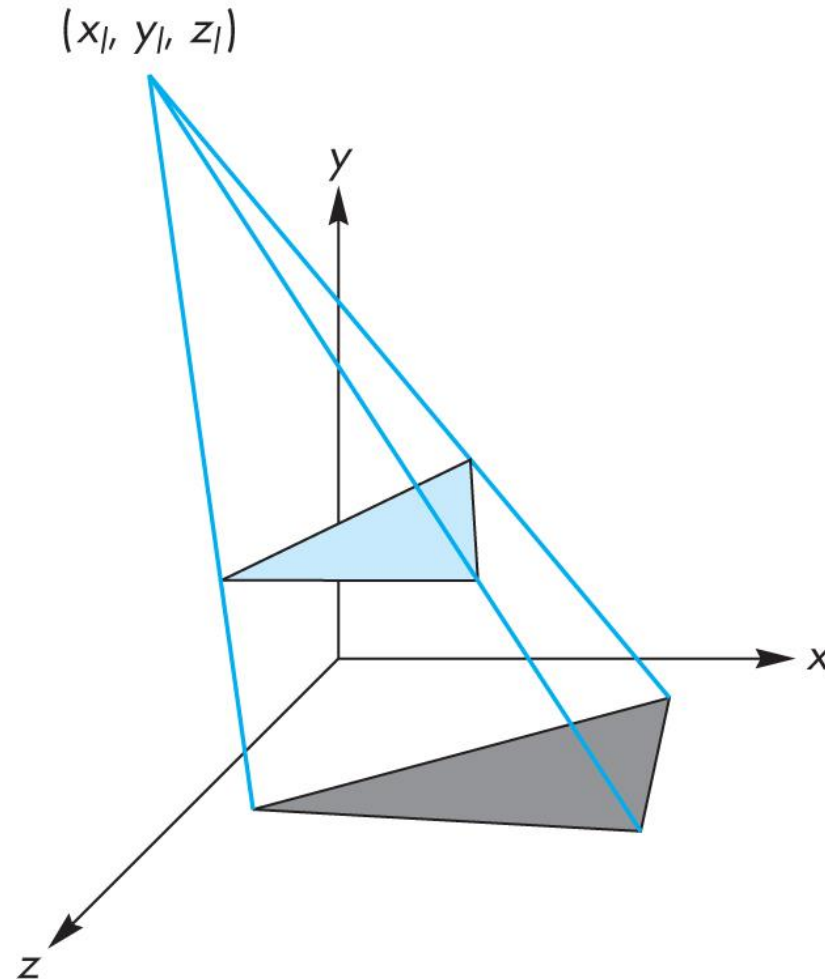
Projective Shadows

- 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



The University of New Mexico

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 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



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 they are treating the texture as a “slide projector”
- This technique is the basis of projective textures and image based lighting
- Supported in desktop OpenGL and GLSL through four dimensional texture coordinates
- Not yet in WebGL



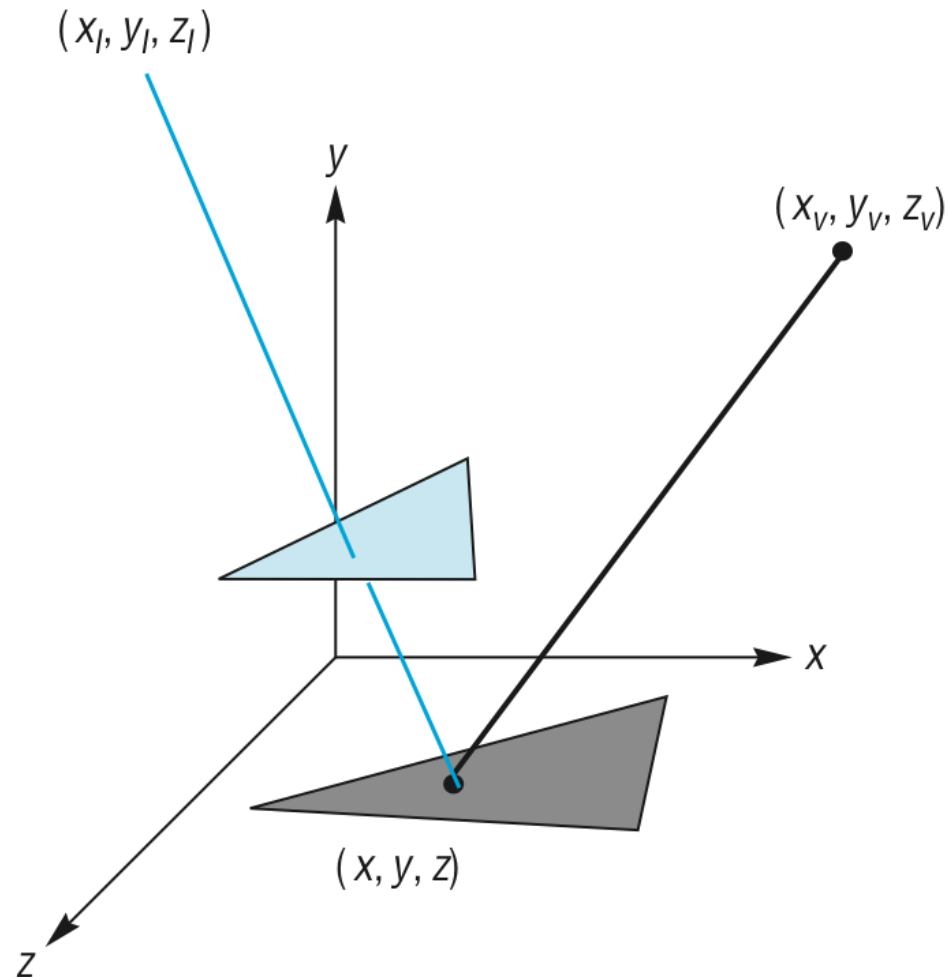
Shadow Maps

- 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



The University of New Mexico

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



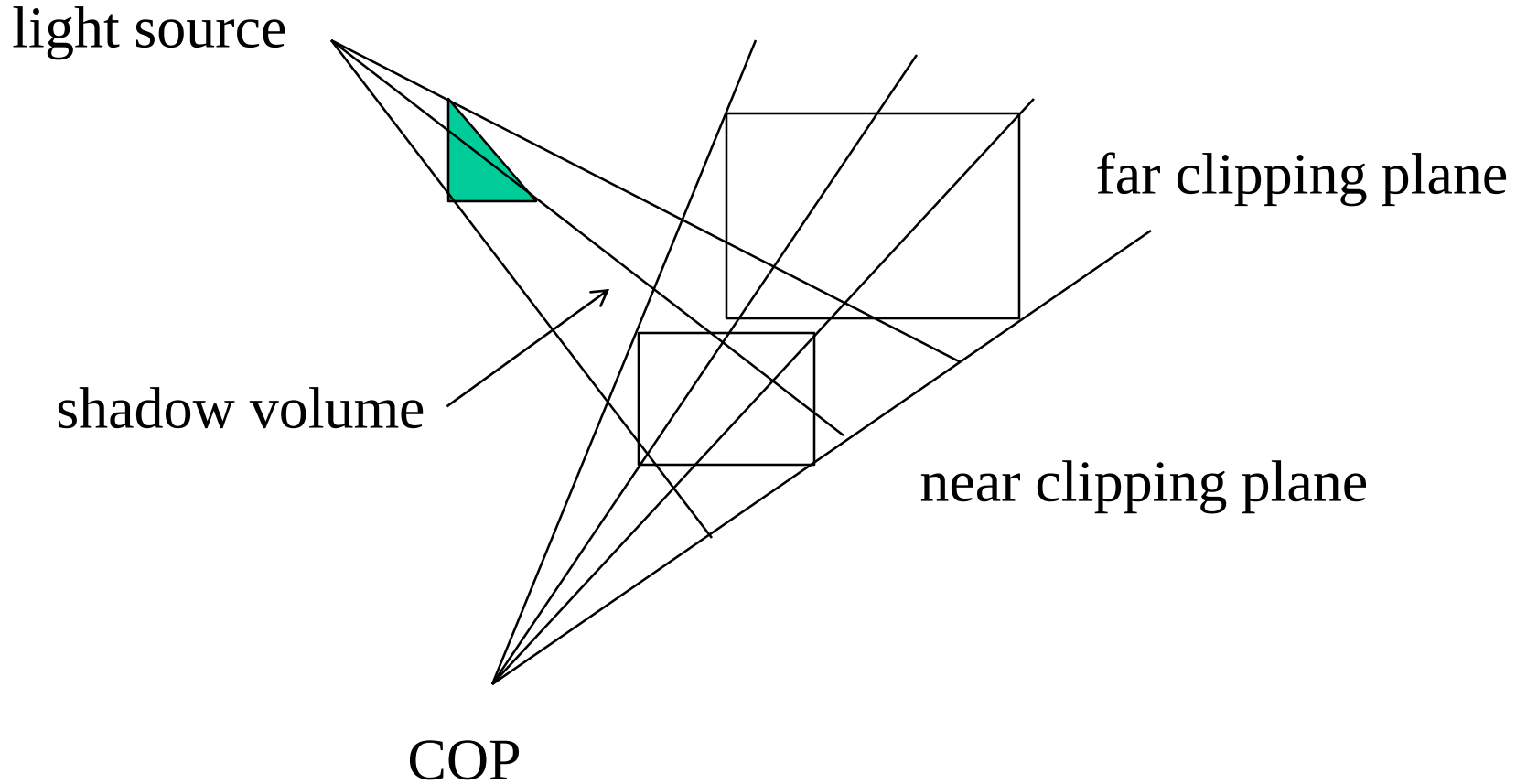
Implementation

- 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



The University of New Mexico

Shadow Volumes





Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research,
Technology and Science Laboratory

University of New Mexico



The University of New Mexico

Lighting and Shading I

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico



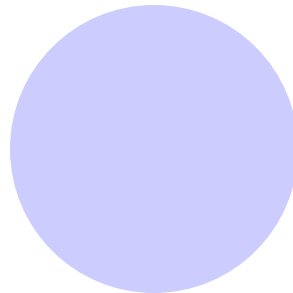
Objectives

- Learn to shade objects so their images appear three-dimensional
- Introduce the types of light-material interactions
- Build a simple reflection model---the Phong model--- that can be used with real time graphics hardware

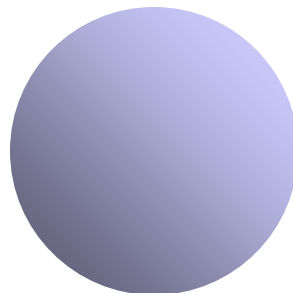


Why we need shading

- Suppose we build a model of a sphere using many polygons and color it with `glColor`. We get something like



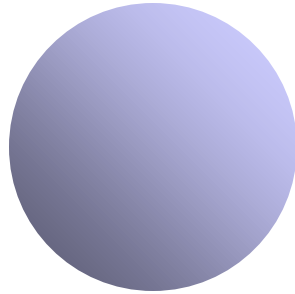
- But we want





Shading

- Why does the image of a real sphere look like

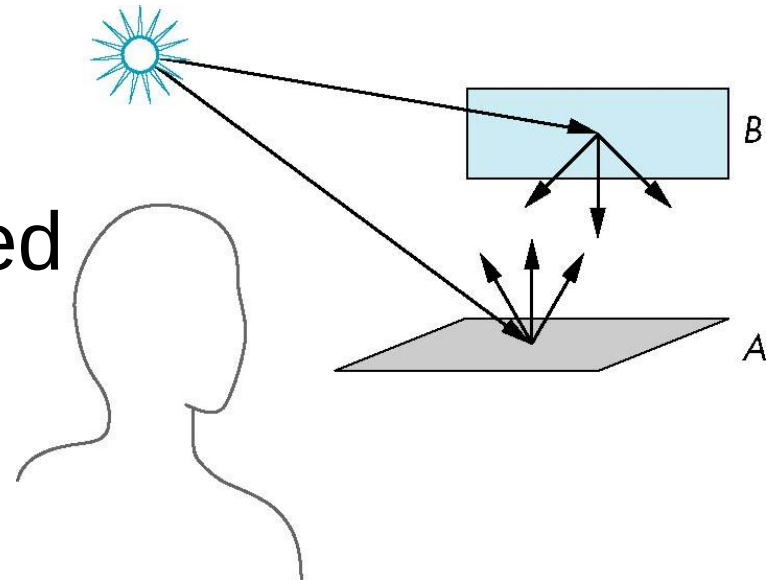


- Light-material interactions cause each point to have a different color or shade
- Need to consider
 - Light sources
 - Material properties
 - Location of viewer
 - Surface orientation



Scattering

- Light strikes A
 - Some scattered
 - Some absorbed
- Some of scattered light strikes B
 - Some scattered
 - Some absorbed
- Some of this scattered light strikes A and so on





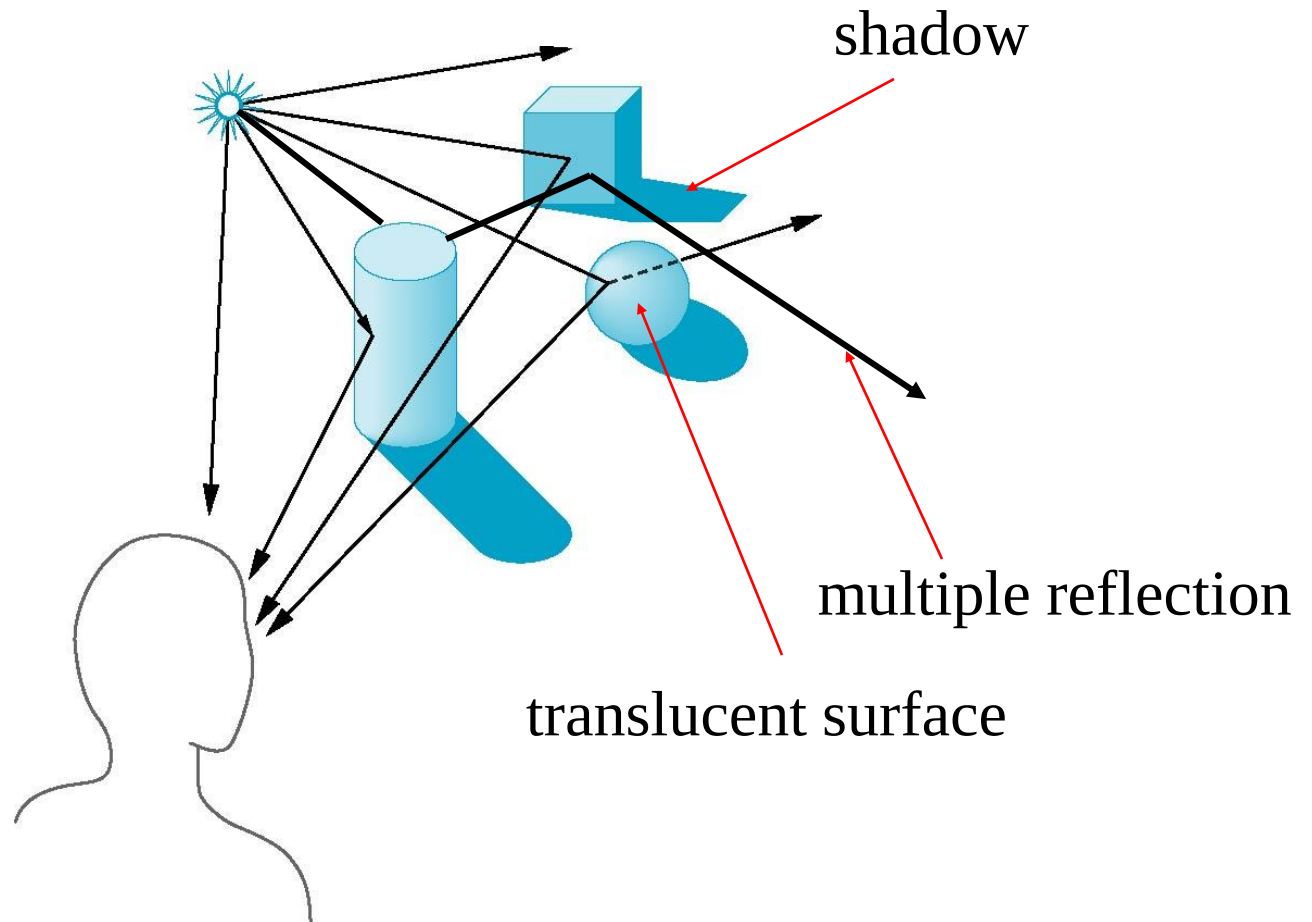
Rendering Equation

- The infinite scattering and absorption of light can be described by the *rendering equation*
 - Cannot be solved in general
 - Ray tracing is a special case for perfectly reflecting surfaces
- Rendering equation is global and includes
 - Shadows
 - Multiple scattering from object to object



The University of New Mexico

Global Effects





Local vs Global Rendering

- Correct shading requires a global calculation involving all objects and light sources
 - Incompatible with pipeline model which shades each polygon independently (local rendering)
- However, in computer graphics, especially real time graphics, we are happy if things “look right”
 - Exist many techniques for approximating global effects



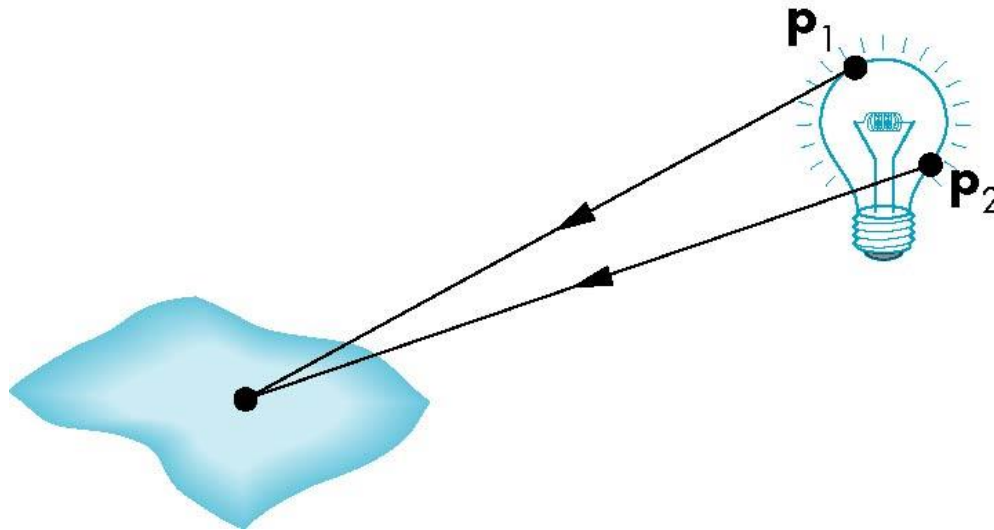
Light-Material Interaction

- Light that strikes an object is partially absorbed and partially scattered (reflected)
- The amount reflected determines the color and brightness of the object
 - A surface appears red under white light because the red component of the light is reflected and the rest is absorbed
- The reflected light is scattered in a manner that depends on the smoothness and orientation of the surface



Light Sources

General light sources are difficult to work with because we must integrate light coming from all points on the source





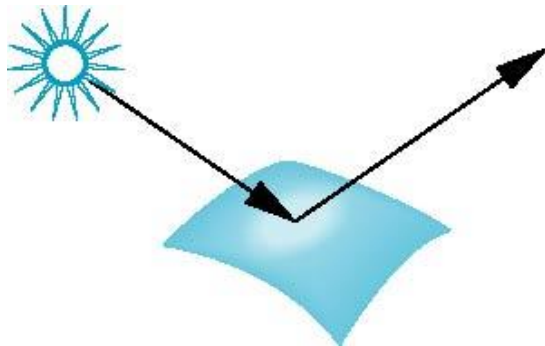
Simple Light Sources

- Point source
 - Model with position and color
 - Distant source = infinite distance away (parallel)
- Spotlight
 - Restrict light from ideal point source
- Ambient light
 - Same amount of light everywhere in scene
 - Can model contribution of many sources and reflecting surfaces

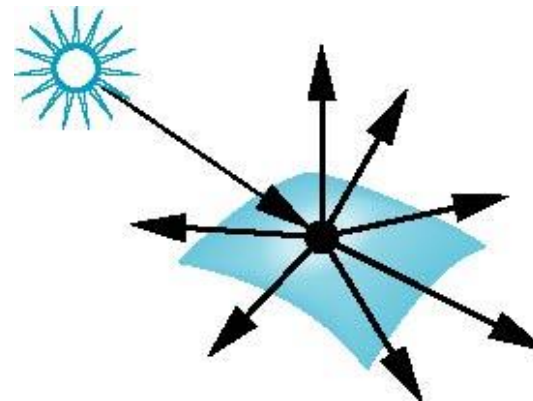


Surface Types

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflect the light
- A very rough surface scatters light in all directions



smooth surface

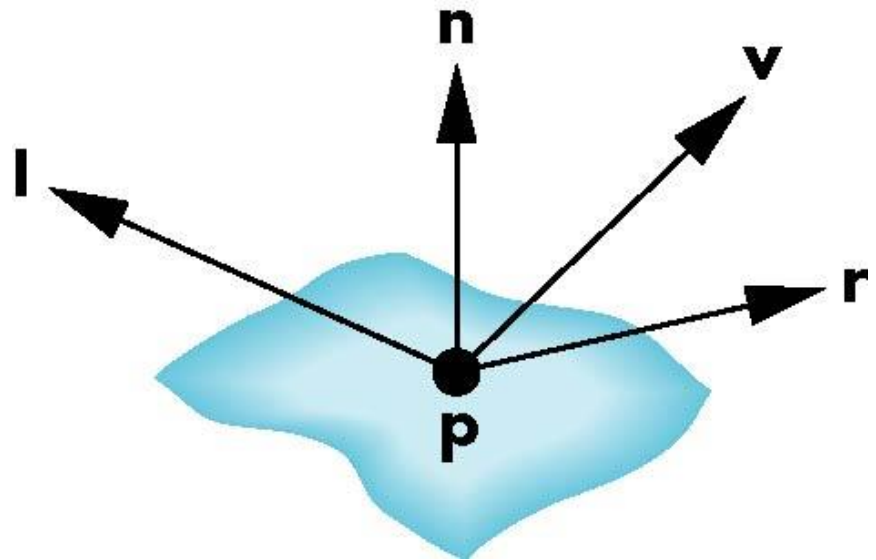


rough surface



Phong Model

- A simple model that can be computed rapidly
- Has three components
 - Diffuse
 - Specular
 - Ambient
- Uses four vectors
 - To source
 - To viewer
 - Normal
 - Perfect reflector

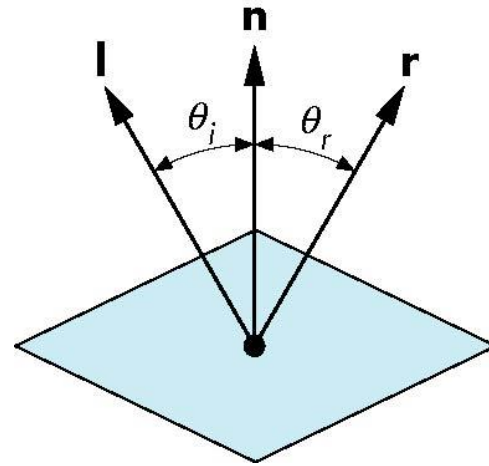




Ideal Reflector

- Normal is determined by local orientation
- Angle of incidence = angle of reflection
- The three vectors must be coplanar

$$\mathbf{r} = 2 (\mathbf{l} \cdot \mathbf{n}) \mathbf{n} - \mathbf{l}$$





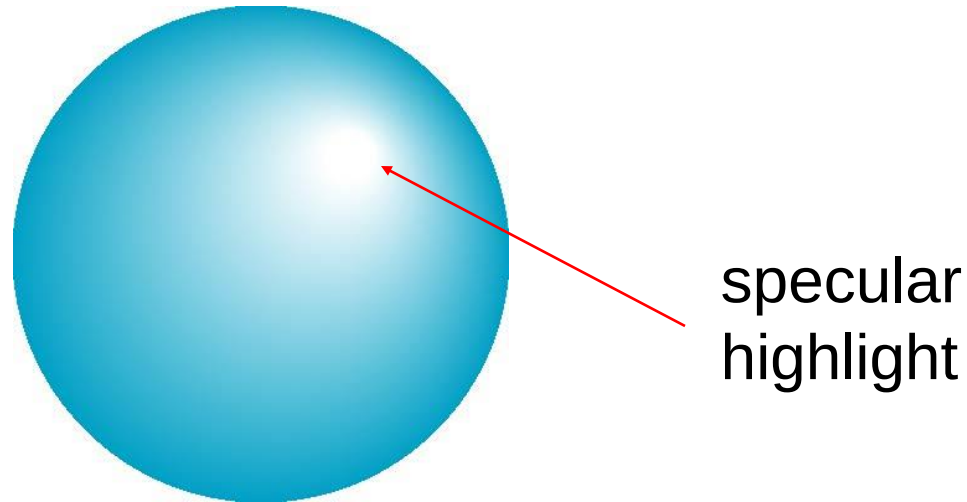
Lambertian Surface

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is proportional to the vertical component of incoming light
 - reflected light $\sim \cos \theta_i$
 - $\cos \theta_i = \mathbf{l} \cdot \mathbf{n}$ if vectors normalized
 - There are also three coefficients, k_r , k_b , k_g that show how much of each color component is reflected



Specular Surfaces

- Most surfaces are neither ideal diffusers nor perfectly specular (ideal reflectors)
- Smooth surfaces show specular highlights due to incoming light being reflected in directions concentrated close to the direction of a perfect reflection





Modeling Specular Reflections

- Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased

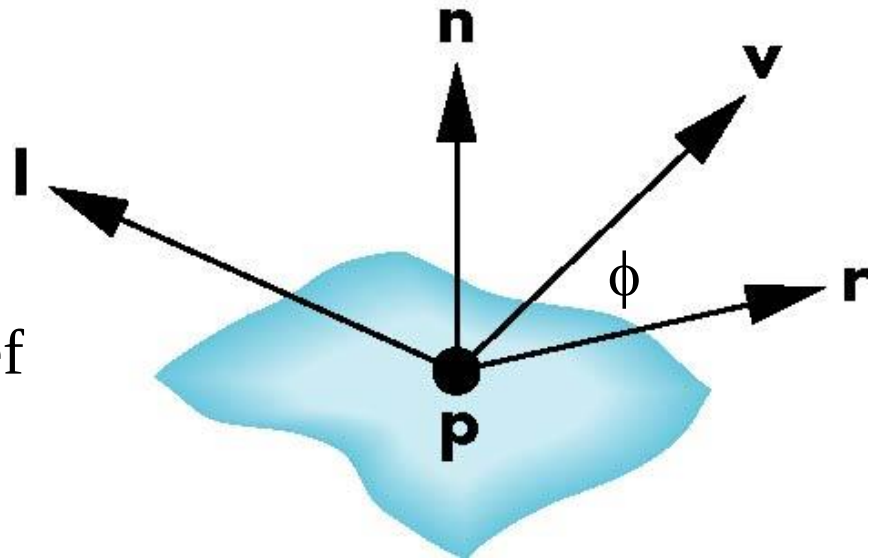
$$I_r \sim k_s I \cos^\alpha \phi$$

reflected intensity

absorption coef

incoming intensity

shininess coef





The University of New Mexico

The Shininess Coefficient

- Values of α between 100 and 200 correspond to metals
- Values between 5 and 10 give surface that look like plastic

