

## CSI 422/502: Lecture 27

Finishing derivation of the perspective projection matrix..  
refer to slide copies handout..  
and new slides below.

If non-zero vector  $\mathbf{V}$  can be written as a linear combination of others from a set of vectors

$$\mathbf{V} = u_1 \mathbf{V}_1 + u_2 \mathbf{V}_2 + \dots$$

we say there is a **linear dependency** of  $\mathbf{V}$  on  $\mathbf{V}_1, \mathbf{V}_2, \dots$ .

A set of vectors is called **linearly independent** means there aren't any linear dependencies among them (the vectors in that set.)

**Greedy algorithm:** Given a (non-zero) linear (vector) space  $S$ , like all 4-tuples  $\{[x, y, z, w]^t : \text{where } x, y, z, w \text{ are numbers}\}$ .

(1) Pick a non-zero vector and call it  $\mathbf{V}_1$ , and set  $i = 1$ .

(2) If you can, pick any other non-zero vector as long as the new vector together with the ones already picked form a linearly independent set. Set  $i = i + 1$  and call the new vector  $\mathbf{V}_i$ .

If you cannot, stop and output  $\{\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_i\}$  and call it a **basis** for  $S$ .

Basic facts of linear algebra:

1. All different bases for the same subspace have the *same number of elements*.  
You will get the *same*  $i$  no matter how you picked  $\{\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_i\}$ .
2. Given a basis  $\{\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_i\}$  every vector in  $\mathbf{V} \in S$  is a linear combination

of basis elements:

$$\mathbf{V} = u_1 \mathbf{V}_1 + u_2 \mathbf{V}_2 + \dots$$

(and the coefficients are unique.)

3. Remember,  $T$  is a **linear transformation** means

$$\text{IF } \mathbf{V} = u_1 \mathbf{V}_1 + u_2 \mathbf{V}_2 + \dots$$

$$\text{THEN } T(\mathbf{V}) = u_1 T(\mathbf{V}_1) + u_2 T(\mathbf{V}_2) + \dots$$

If you know  $T(\mathbf{V}_1)$ ,  $T(\mathbf{V}_2)$ ,  $\dots$ ,  $T(\mathbf{V}_i)$  then you can determine the value  $T(\mathbf{V})$  for ANY, ARBITRARY vector  $\mathbf{V}$  in the subspace.

We will use this “fundamental theorem” to figure out the projection matrix from the information about what it should do to a few homogeneous coordinate vectors.

Equation of a plane (homogeneous coords):

$$\begin{pmatrix} A & B & C & D \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = Ax + By + Cz + Dw = 0$$

For those who attended Prof. Lomonoco's lecture:

He called a vector like  $P$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \text{ a "Ket" and wrote it } |P\rangle.$$

He called a vector used as plane equation coefficients like

$$L = \begin{pmatrix} A & B & C & D \end{pmatrix} \text{ a "Bra" and wrote it } \langle L|.$$

Then the "dot product", the substitution of point coordinates into the plane equation's left hand side, is denoted:

$$\langle L | P \rangle \text{ and is called a Bracket (=Bra+ket)}$$

1. Plane is a **projection surface** for perspective or parallel projection.

2. Plane is one of the 6 NDC clipping planes:

$$x + w = 0; -x + w = 0; y + w = 0 - y + w = 0 z + w = 0 - z + w = 0$$

3. Plane is one of OpenGL **optional clipping planes**; see HB 7-12 or RedBook.

Good for section views.

OpenGL requires at least 6 be accommodated, allows implementations with more.

4. Plane is part of a **model for light reflection, scattering, coloring, emission, etc.**

Each OpenGL **normal vector** (attached, like 3D coordinates and RGBA color, to each OpenGL **vertex** (generated by `glVertex*()` operations)) is the  $(A, B, C)$  of the plane equation.

5. OpenGL supports drawing of **surfaces with triangle**, (planar) **quadrilateral** and (planar) **polygon** primitives.

Higher level software:

- (a) Typically has mesh data structures, and hierarchical world models using them, for a 3D world of solid objects, lights, cameras, etc.
- (b) Might generate the polygons for a surface one by one, calculating the **normal vector** and **vertex coordinates** for each.

See Redbook's **Building an Icosahedron Example**.

The vector **cross product** and length calculation is used to calculate the normal vectors.

*Important question “Issue” in 3D: Which side of a (planar) polygon*

1. faces the viewing direction??
2. faces the outside of a (mesh represented) surface which the polygon is part of?

Reasons:

1. The two different sides might have different colors.
2. If the polygon is part of a “closed surface” (actually a 2D surface without any 1D boundary edges), none of the inside will be visible from the outside. So, all rendering work can be avoided if the *outside* side faces *away from* the viewer.

Saving work this way is called **backface culling**.

But backface culling should not be done if we sometimes want to see the inside of the surface. (Blender training HW shows this.)

In 2D, given a polygon (possibly with crossing sides), which regions are **inside** and which are **outside**?

How to calculate the normal for a planar surface:

**Way 1:** Given coordinates of 3 points  $A[x_a, y_a, z_a]$ ,  $B[x_b, y_b, z_b]$ ,  $C[x_c, y_c, z_c]$ , not all in one line,

calculate  $B - A = [x_b - x_a, y_b - y_a, z_b - z_a]$  and  $C - A = [x_c - x_a, y_c - y_a, z_c - z_a]$  then calculate  $K = (B - A) \times (C - A)$ , and “normalize” by calculating  $|K|$  and dividing (3 times) to get  $N = \frac{K}{|K|}$ .

The resulting vector will point **TOWARD** a viewer who sees the sequence  $A, B, C$  traversing the triangle in **positive or COUNTER-CLOCKWISE** order.

Mesh models will typically use the order of the vertices in a polygon to determine which side is the **OUTSIDE**.

**Way 2:** (Really cool way to use projective geometry to remember the formula for a plane equation in terms of 3 points in the plane).

The equation of the plane determined by 3 non-collinear points  $A, B, C$  is

$$\det \begin{bmatrix} x & x_a & x_b & x_c \\ y & y_a & y_b & y_c \\ z & z_a & z_b & z_c \\ 1 & 1 & 1 & 1 \end{bmatrix} = 0$$

This is one scalar equation

$$\alpha x + \beta y + \gamma z + \delta = 0$$

where

$$\alpha = \det \begin{bmatrix} y_a & y_b & y_c \\ z_a & z_b & z_c \\ 1 & 1 & 1 \end{bmatrix}$$

Way (3): What if the mesh's polygon  $P$  has 4 or more vertices, and they are not quite all in the same plane??

Foley, Van Dam, Feiner, Hughes' big *Computer Graphics Principles and Practice* book says:  $(\alpha, \beta, \gamma)$  are proportional to the 3 areas of the projections of  $P$  on the  $yz$ ,  $xz$  and  $xy$  planes!

Calculation of normals to be attached to OpenGL vertices is necessary for OpenGL shading.

Flat shading: the one normal belonging to the LAST polygon vertex is used.

READ HB 10-1, 10-2, 10-3, 10-20 pages 637-647; skim 10-10 for now.

Blender Homework for lighting and shading.

HB's vocabulary:

**illumination or lighting model:** for calculating the light intensity at each point on a surface.

**surface rendering:** given the light intensity, plus the normal, color and other properties at a surface point, plus distance and direction to the viewer, calculate what RGB value to color the corresponding pixel.

Models *simplify* physics, physiology, and psychology. (They can generate non-realistic images that are attractive and/or informative.)

Factors variously used in various models:

1. Material properties: Opacity, transparency, shiny or dullness, color (different colors can be specified for different model aspects: e.g., **ambient light**, **diffuse light** and **specular light** surface texture: patterns and/or bumpy roughness.
2. Light source features: shape (of emitting surface and/or angular distribution), color, intensity, distance, position.

3. Viewing features: Distance from surface, fogginess, ...
4. Geometric relationships: Angles among (1) surface normal vector, (2) direction to a light source and (3) direction to the viewer.

**Lambert or diffuse reflection:** Surface appears *equally bright from all directions* because whatever light is incident on the surface is reflected *equally in all directions*

Dull, matte surfaces such as chalk or sand reflect light this way. Sand is composed of irregular, sphere-like particles.

So the **intensity** (average power over area, brightness of **ONE SPOT**) of the diffuse reflected light depends **ONLY** on the intensity of the incident light:

Ambient Light: no directionality, Lambert surfaces look flat.

Directional Light: Lambert surfaces show dull highlights.

*Diffuse reflection of a non-directional or ambient light source:*

All surfaces are illuminated the same way independently of the surface orientation. (The surface orientation is perpendicular to the surface normal.)

The image has the same intensity no matter what angle is between the normal and the viewer.

*Diffuse reflection of a directional light source:* The light comes from a particular direction  $\mathbf{L}$ . The intensity at the surface is proportional to the cosine of the angle between  $\mathbf{L}$  and the surface normal  $\mathbf{N}$

The reflection has the same intensity independent of viewing angle, (just like ambient lighting of the diffuse reflector.)

So, the image intensity is independent of viewing direction, but is proportional to angle between the surface normal and the light direction.

*Specular reflection of a directional light source:* Vector  $\mathbf{L}$  is the direction from the surface point to the light. The reflection intensity varies with all the  $\mathbf{L}$ ,  $\mathbf{N}$  and direction  $\mathbf{V}$  to viewer: It is most intense in the direction  $\mathbf{R}$  given by Snell's law for perfect mirrors:

The “angle of reflection” (away from the normal in the same plane as the normal and incident light ray) equals the “angle of incidence” (away from the normal in the same plane, in the opposite direction).

The **Phong model** is one of several (shading models) used to calculate the image intensity from  $L$ ,  $N$ , and  $V$  together with parameters. It is a compromise between efficiency and realism.

Some physics....

An **extensive** or “bulk” quantity pertains to an extended object. Examples: Volume, Weight, Mass, Gross National Product, Area, Length, Electric Charge, Energy, Power ...

Light Energy emitted, reflected or absorbed by a **GIVEN AREA** during a **PARTICULAR PERIOD** of time.

Light Power (=Energy/Time) is the rate of Light Energy transfer.

(Electric energy is sometimes measured in Kilowatt-hours. Food energy is measured in Calories.

Electric power is usually measured in Kilowatts or Watts. Food (people) power is measured in Calories per Day.)

An **intensive** or “local” quantity pertains to a point in an object or an average over space. Examples: Density = Mass/Volume; Average National Income.

Light Intensity is Light Power per unit Area.

The Light Intensity of light reflected toward the viewer determines the **BRIGHTNESS** of one **SPOT** in an image.

In 2D, the answer is defined using the **winding number** at a “*test point*” of the polygon.

The practical, OpenGL Solution:

Only use polygons (like non-degenerate triangles) that are  
**planar: all vertices in one plane.**

**simple: edges never cross or touch except for adjacent edges at their endpoints.**

**and convex: For every pair of points  $P_1$ ,  $P_2$  inside or on the boundary of the polygon, the *entire line segment* between  $P_1$  and  $P_2$  is part of the polygon.**

(A convex polygon has no “cavities” or “harbors” or “indentations”.)

If an application generates models with non-convex polygons as faces, it must **tessellate** each non-convex face before asking OpenGL to render it.

See Redbook Chapter 11:

GLU (OpenGL standard utility library ON TOP OF, but not INSIDE OpenGL) has support for tessellating non-convex (i.e., concave) polygons.

Also, GLU support for defining curved surfaces and lines (like circles) by quadric formulas (degree 2 polynomials) and then generating lots of polygons or straight lines for OpenGL to approximate them.

Clipping Algorithms for line segments is 2D or 3D:

Input: The 6 clipping planes and the coordinates of the 2 endpoint vertices of a line segment, all in NDC.

**Cohen-Sutherland algorithm** (1) Calculate two **region codes**, one for each endpoint, based the 9 or 27 regions the 2 or 3 pairs of clipping planes determine.

(2) Use non-floating point logical calculations to rule out segments that lie entirely on the outside side of any one clipping plane.

(3) Use a (floating point) line-plane intersection calculation for the hard cases.

## **Liang-Barsky algorithm:**