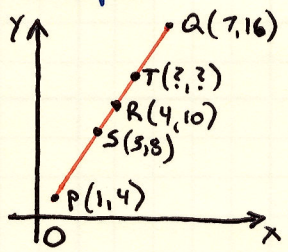
Outcomes

- Understand how and why interpolation is used in CG
- Understand the importance of convexivity
- Understand the concepts of Triangulation and how it is used
- Understand Orientation and its relationship to Transformation and Triangulation

Convexity and Interpolation

- We know that OpenGL favors 3 basic primatives: points, line segments and triangles, this is due to:
  - Every point inside these primatives can be uniquely represented in terms of its vertices
  - This makes it possible for properties, such as color, to be programmatically interpolated throughout the primitive

- Example



Draw the segment PQ joining the point P(1,4) to the point Q(7,16). Measure off the mid point R of the segment. Verify the coordinates are as shown in the figure. Since the midpoint is halfway from either endpoint it does make sense that the coordinates of R are an exact average of P and Q:

$$\frac{1}{2}\cdot(1,4)+\frac{1}{2}\cdot(7,16)=\left(\frac{1}{2}\cdot 1+\frac{1}{2}\cdot 7,\ \frac{1}{2}\cdot 4+\frac{1}{2}\cdot 16\right)=(4,10)$$

- Exercise: Calculate the coordinates of T which is $2/5$ of the way from P to Q

- General Expression: $X = c*(x_1,y_1)+(1-c)*(x_2,y_2)$

- Exercise: If the end vertex P of the segment in the preceeding example is specified red (RGB=(1,0,0)) and Q green (RGB=(0,1,0)), then what should the color be at the mid point R? at T?

° Convex Combinations

- All polygons in OpenGL are first sub-divided into triangles prior to being processed. This is due to the fact that points, segments, and triangles have the property of being uniquely interpolate for any point in the primitive

- Segments:

  ° Prop 7.1: if P and Q are two points in $\mathbb{R}^3$, then a point V lies on the segment PQ if and only if it can be expressed as

  $$V = c_1 P + c_2 Q$$

  where $0 \le c_i \le 1$, for both $i=1$ and $i=2$, and where $c_1 + c_2 = 1$

  Further, if P and Q are distinct – so that PQ does not degenerate to a point – then this expression for V is unique.

  ° Thus the equation for a point V is:

  $$(v_x, v_y, v_z) = c_1(p_x, p_y, p_z) + c_2(q_x, q_y, q_z) = (c_1 p_x + c_2 q_x, c_1 p_y + c_2 q_y, c_1 p_z + c_2 q_z)$$

  - Example: $P(1,4,3)$ $Q = (2,5,2)$, then points on segment PQ are of the form

    $(c_1 + 2c_2, 4c_1 + 5c_2, 3c_1 + 2c_2)$, where $0 \le c_1, c_2 \le 1$ and $c_1 + c_2 = 1$

  ~~Thus~~

  ° This can be simplified to the following:

  A point V of the form

  $$V = c_1 P + c_2 Q$$

  where $0 \le c_i \le 1$, for both $i=1$ and $i=2$, and where $c_1 + c_2 = 1$ is said to be a convex combination – or barycentric combination – of P and Q. The scalars c_1 and c_2 are called the barycentric coordinates of V.

  ° Corollary 7.1: The segment joining two points P and Q in $\mathbb{R}^3$ consists of all their convex combinations

  - A point $V = c_1 P + c_2 Q$ can be thought of as a weighted sum of P and Q

- **Triangles**

Prop 7.2: If P, Q, and R are three points in $\mathbb{R}^3$, then a point V lies on the triangle PQR iff it can be expressed as

$$V = c_1 P + c_2 Q + c_3 R$$

where $0 \leq c_i \leq 1$, for $1 \leq i \leq 3$, and where $\sum c_i = 1$

This expression is unique if PQR is not collinear

This combination is called the convex combination - or barycentric combination - of P, Q, and R. The scalars c_i are called the barycentric coordinates of V

Cordlary 7.2: The triangle w/ vertices at P, Q, R in $\mathbb{R}^3$ consists of all their convex combinations.

- **Example:** If $P = (0,0,0)$, $Q = (20,0,0)$ and $R = (20,30,0)$, does the point $V = (10,20,0)$ lie on the triangle PQR? If so, express V as a convex combination of the three vertices.

$$V = c_1 P + c_2 Q + c_3 R$$

$$c_1 + c_2 + c_3 = 1$$

$$(10,20,0) = c_1(0,0,0) + c_2(20,0,0) + c_3(20,30,0)$$

$$20c_2 + 20c_3 = 10$$
$$30c_3 = 20$$
$$c_1 + c_2 + c_3 = 1$$

$$\therefore \; c_1 = \frac{1}{2} \quad c_2 = -\frac{1}{6} \quad c_3 = \frac{2}{3}$$

As the c_i's do not all lie between 0 and 1 we conclude V is not a convex combination of P, Q, and R and thus V does not lie on ΔPQR

Interpolation

- Let the RGB color triples at points P_1, P_2, P_3 be (R_1, G_1, B_1), (R_2, G_2, B_2), (R_3, G_3, B_3) and let point V be specified as follows:

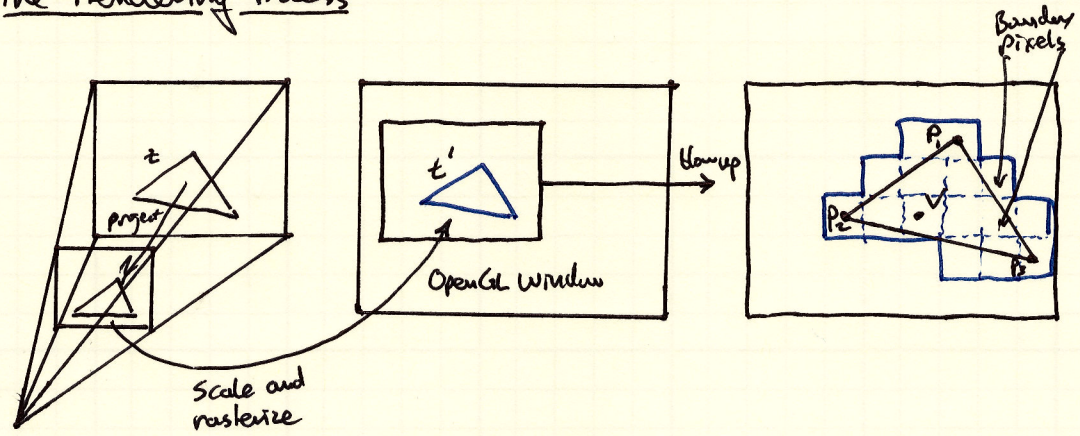
$$V = c_1 P_1 + c_2 P_2 + c_3 P_3$$

- Then the color of V is specified as

$$c_1(R_1, G_1, B_1) + c_2(R_2, G_2, B_2) + c_3(R_3, G_3, B_3)$$

$$= (c_1 R_1 + c_2 R_2 + c_3 R_3,\ c_1 G_1 + c_2 G_2 + c_3 G_3,\ c_1 B_1 + c_2 B_2 + c_3 B_3)$$

- As the weights c_i are unique, the interpolation is unambiguous and programmable.
- This process of interpolation is called smooth shading or Gouraud shading
- The Rendering Process



  - A practical application of interpolation to rendering must take into account the fact that screen space is not actually a 2D continuum but a rectangular array, called a raster, of finitely many pixels.
  - Based on the shoot-and-print model of rendering, a primitive object, such as a triangle t, drawn in the viewing volume is projected to the volume's front and, then, scaled to its image t' on the OpenGL window (see above)
  - The image t' is what we see and is actually rendered as a set of pixels.
  - The part of the process consisting of choosing and coloring the pixels to render t' is called rasterization or scan conversion

- Rasterization
  - A pixel is a square that contains not one point, but infinitely many
  - This requires OpenGL to pick a representative one at which to interpolate the color values from the vertices and then set the entire pixel RGB to those values
    - An interior pixel should use the object's center point, ~~the color~~ V_p. The color of the pixel is set by:

$$c_1(R_1, G_1, B_1) + c_2(R_2, G_2, B_2) + c_3(R_3, G_3, B_3)$$

where
$V = c_1P_1 + c_2P_2 + c_3P_3$ and the specified color at P_i is (R_i, G_i, B_i) for $1 \leq i \leq 3$

    - Boundary pixels are more complicated as we must take into account both foreground and background colors and any other effects, such as anti-aliasing.
- Finally, we know that points, segments, and triangles can have their properties interpolated. Does this apply to polygons w/more than 3 vertices?
  - In short, No!

Convexity and the Convex Hull

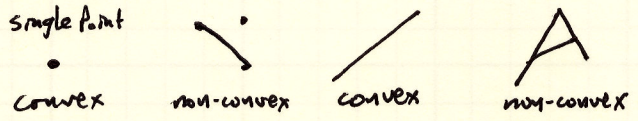
- Convex combinations can be defined for an arbitrary number of points:

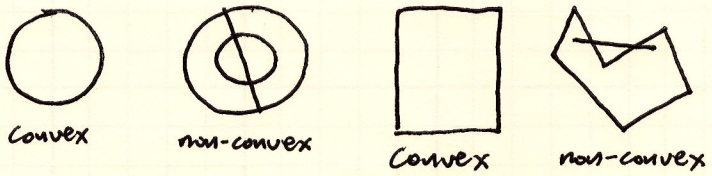
  If $F = \{P_1, P_2, \ldots, P_k\}$ is a set of k points, then a point V of the form

  $$V = c_1P_1 + c_2P_2 + \ldots + c_kP_k$$

  where $0 \leq c_i \leq 1$, for $1 \leq i \leq k$, and where $c_1 + c_2 + \ldots + c_k = 1$, is said to be a convex combination of F. The scalars c_i, $1 \leq i \leq k$, are the barycentric coordinates of V

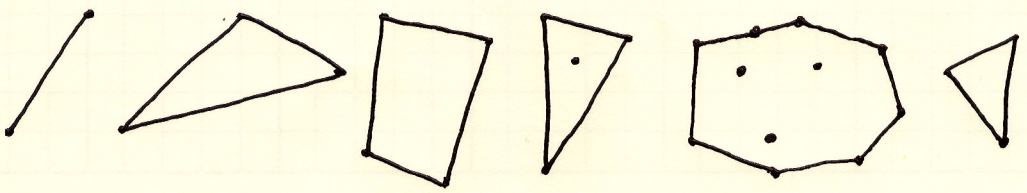
- A set S of points is said to be convex if, for any two points $P, Q \in S$, it is true that the segment $PQ \subset S$; in other words, if it is true that, if the end points of a segment are in S, then the segment itself is contained in S





- Consider the collection of all convex sets containing a given set F.
  - This collection contains the plane, and possibly, many other sets
  - The intersection, X, of this collect surely contains F as each member does, and X is convex as well.
  - X is no larger than convex set C containing F, thus there always exists a smallest convex set containing a given planar set F: it is simply the intersection of all convex sets containing F

- The smallest set containing a given set F on the plane is called its convex hull, denoted $ch(F)$
  - Obviously, a set F is convex iff it is its own convex hull, i.e., iff $F = ch(F)$



- Prop 7.3: Given a set $F = \{P_1, P_2, \ldots, P_k\}$ of k points in $\mathbb{R}^2$, $ch(F)$ is exactly the set of convex combinations of F.

- Extreme Points

  - If a point P of a set F of points on the plane is such that the convex hulls of F and $F - \{P\}$ (F with P deleted) are the same, the P is said to be a non-extreme point of; otherwise, it is an extreme point of F.

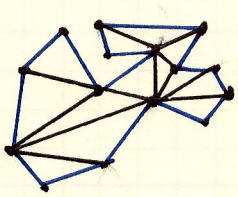
  - Non-extreme points are expendable and thus there removal has no effect on the convex-hull

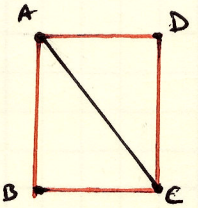
  - Removing an extreme point makes the convex hull smaller

[redacted]

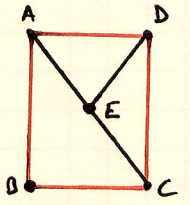
Triangulation

- 3D objects are assembled as a surface composed of 2D primitives
- Specifically as a combined mesh of triangles
- The combination or decomposition into this set of triangles is done through triangulation.
- Definition: Suppose $\mathcal{T}$ is a collection of triangles whose union is an object X. Then $\mathcal{T}$ is said to be a triangulation of X if, given any two different triangles t_1 and t_2 from $\mathcal{T}$, exactly on of the following three is true:
  1. t_1 and t_2 are disjoint, i.e., do not intersect at all
  2. t_1 and t_2 intersect in a vertex of both
  3. t_1 and t_2 intersect in an edge of both
- If $\mathcal{T}$ is a triangulation of X then X is said to be triangulated by $\mathcal{T}$
- A collection of triangles may be such that its union is X without being a triangulation of X (according to the above rules) – This is called an invalid triangulation.

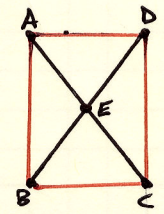




{ABC, ADC}
Triangulation

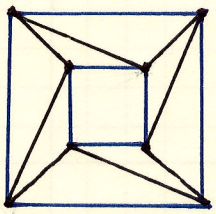


{ABC, DAE, DEC}
Invalid Triangulation

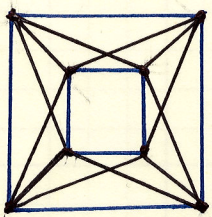


{ABE, BCE, CDE, DAE}
Triangulation

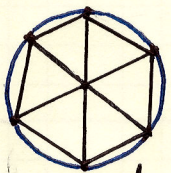
{ABC, DBE, DAE}
Invalid Triangulation



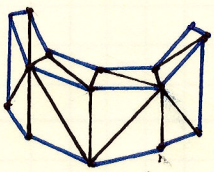
Triangulation



Invalid Triangulation



Triangulated Polygon



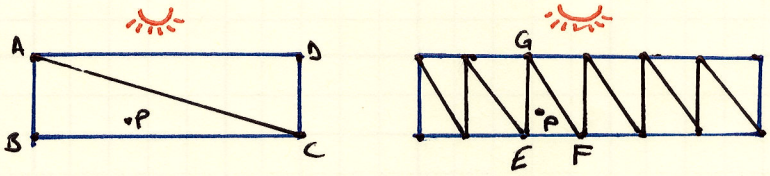
Triangulated Wall

- Triangulation is not a unique process, thus the same object may have many triangulations.

- In practice, ambiguity of triangulation is resolved, regardless of code order, by the polygon rasterizing algorithm.
- Prop 8.1: If the collection of triangles composing an object satisfies the properties of a triangulation, then its image is independent of the order in which the triangles are rendered, i.e., independent of code order.

Steiner Vertices and Triangulation Quality

- Steiner Vertex ≡ a vertex used in the triangulation of an object which is not a vertex of that object
- Steiner vertices are often used to improve the quality of triangulation
  - A good triangulation is one with few long and thin triangles, called slivers, and where most triangles are of nearly equal size and relatively small, with respect to the object.



  - In the example above the long sliver ABC causes what should be local brightness at P to be global across the object, while the intensity of P in EFG is more realistic.

Orientation

- For each triangle the viewer currently sees, OpenGL must determine which side is visible
- Why is this?
  - Each side has properties (outlined/filled, color, etc.) specified differently and OpenGL is obliged to display accordingly

How OpenGL Determines Back + Front Faces

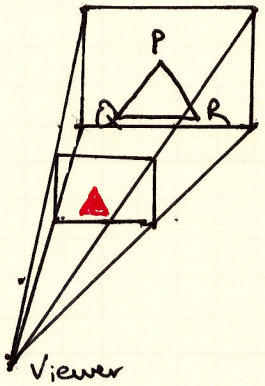
- 1.) Obtains the vertex orders of each 2D primative from the code

Ex:
```
glBegin(TRIANGLES);
    V0; V1; V2; V3; V4; V5;
glEnd();
```
- V0, V1, V2
- V3, V4, V5

Ex:
```
glBegin(GL_TRIANGLE_STRIP);
    V0; V1; V2; V3; V4; V5;
glEnd();
```
- V0, V1, V2
- V1, V3, V2
- V2, V3, V4
- V3, V5, V4

2.) For each triangle determine if it is perceived as CW or CCW by the viewer. This is said to be the orientation of the triangle w.r.t the viewer.

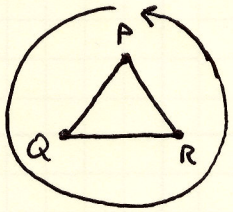
Ex:



PQR is the order

and thus is perceived by the

viewer as CCW



3.) Those triangles perceived to be oriented CCW are presumed to be front-facing, while those perceived to be oriented CW are presumed to be back-facing

- This presumption can be changed with glFrontFace(GL_CW)

- This process is required because OpenGL renders an object a triangle at a time, and may not render the whole object. Regardless, the renderer does not have the whole view of the object and thus cannot identify the absolute faces that will be seen by the viewer.

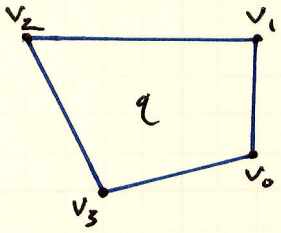
- This process then provides the necessary programmer guidance to OpenGL.

- Two orders of the vertices of a polygon are said to be equivalent if one can by cyclically rotated into the other.

  - Example:

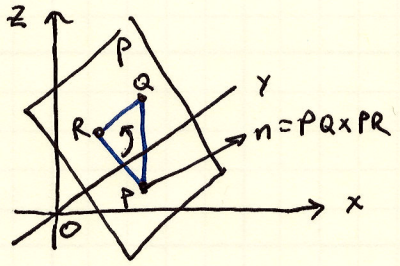
  CCW equivalent: $v_0v_1v_2v_3$ $v_1v_2v_3v_0$ $v_2v_3v_0v_1$ $v_3v_0v_1v_2$

  CW equivalent: $v_0v_3v_2v_1$ $v_3v_2v_1v_0$ $v_2v_1v_0v_3$ $v_1v_0v_3v_2$



- Algorithm to Decide the Orientation Perceived by a Viewer

  - Assume that the viewpoint is at the origin O and that the vertices of a triangle are $P=(x_1,y_1,z_1)$, $Q=(x_2,y_2,z_2)$, and $R=(x_3,y_3,z_3)$. Does the viewe perceive the order PQR as CCW or CW



2.) Cross-Product Approach
the normal of P, n, is the cross product:
$n = PQ \times PR$
if the angle between n and segment $PO < 90°$
then ΔPQR is CCW
else CW

1.) Approach using the determinant

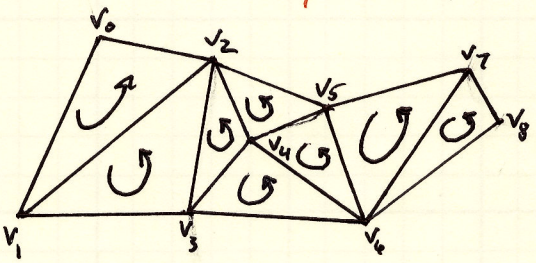
Let D be the determinant of

$$\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{vmatrix}$$

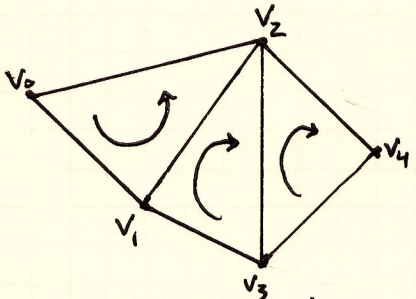
1.) if $D=0$
   a.) PQR are collinear and ΔPQR is degenerate, thus orientation is moot
   b.) O is on plane p and thus sees ΔPQR edge on and orientation is moot
2.) if $D>0$, viewer perceives PQR as CW
3.) if $D<0$, viewer perceives PQR as CCW

Consistently Oriented Triangulation

- When considering a triangulation, the issue of consistency comes up
- Suppose an order is given of the vertices of each triangle belonging to some triangulation T of an object X. T is said to be consistently oriented if any two triangles of T which share an edge order that edge oppositely; otherwise, T is inconsistently oriented



Consistently-oriented triangulation



Inconsistently-oriented triangulation

- Here the 2nd figure is inconsistently oriented, as the edge shared by the left two triangles is ordered $v_1 v_2$ by both
- An observer is not relegated to being able to only view one side of a consistently oriented surface, but rather sees a change in side across boundary edges but never across an internal edge.
- Note: It is a common error to mix-up orientation when assembling an object from multiple pieces.
- Non-Orientable Surfaces
  - There do exist surfaces which can be triangulated but never consistently oriented.
  - Two famous examples are:
    - Mobius Band
    - Klein Bottle
  - Such surfaces are said to be non-orientable.
  - Surfaces for which consistently oriented triangulations do exist are orientable
  - Further formalization requires study of Topology

Culling Obscured Faces

- A surface that bounds a solid is called a closed surface.
- If such a surface is opaque then only one side is visible, a viewer on the outside may only see the outside.
- For the external viewer all back faces are on the outside, and hidden
- For the internal viewer the precise opposite is true.
- Unfortunately, OpenGL has no way of knowing if a surface is closed while rendering. Thus, without extra direction it process each and every triangle and both sides.
- In either of these situations, if we know the position of the viewer we can make OpenGL more efficient by preventing the processing of back-faces
  - This process is called back-face culling or polygon culling
  - This is enabled in OpenGL via:

    ```
    glEnable(GL_CULL_FACE)
    ```

  - And disabled with

    ```
    glDisable(GL_CULL_FACE)
    ```

  - The command glCullFace(face) where face can be GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK is used to specify if front faces, back faces, or all faces are to be removed, respectively.

Transformations and Orientation

- Recall Rigid Transformations (orientation-preserving Euclidean transformations)
  - These preserve the viewer's perceived orientation of a primitive
- In OpenGL, the default is that if a primitive's vertex order is CCW, the viewer is shown the front
  - Reflection about the yz-plane (a non-orientation preserving transformation) will flip all perceived orientations.
  - To prevent this flipping of orientations we can revise our viewing agreement with a call to glFrontFace(...)

For Next Time

- Read Ch 10
- Review Prior Lectures
- Continue working on HW 02
- Come to Class

Additional Notes