

Polygons, State Management and Animation

January 21st 2009

MAE 574, Virtual Reality Applications

Instructor: Govindarajan

Srimathveeravalli

Updates on HW1

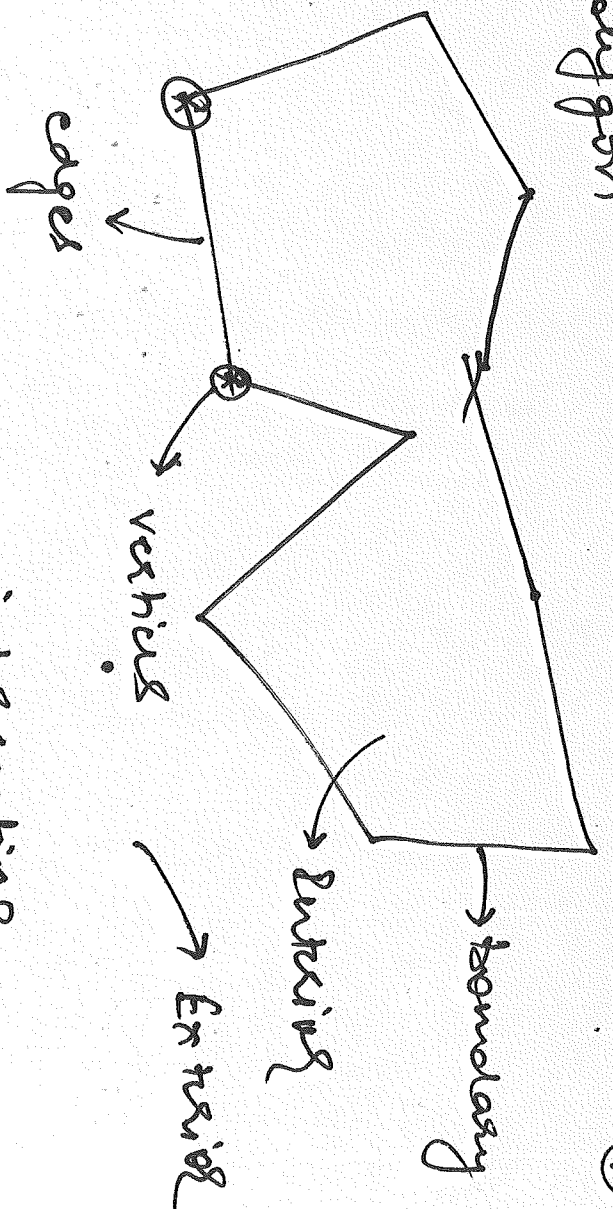
- Make sure to include lines to represent the “coordinate axis”
- Make sure to “synchronize” gluOrtho and window to make sure line is drawn properly
- Make sure the draw and reshape functions are done correctly to ensure that the drawing stays on screen on does not get erased
- Submission
 - Due Friday, before 11 PM
 - Email source code and any additional documentation to my email id.
 - Please make sure to add as many comments as possible to provide maximum information to understand your project.

Polygons - definition

- Definition
 - edge and vertices
 - Interior, exterior and boundary
 - Simple and non simple
 - Convexity
 - Sum of internal angle
 - Non intersection of diagonal and edge
 - Monotone
 - What about “cyclic” and “equilateral”?

A Simple Polygon

- how
- edges
- vertices
- normals
- ?

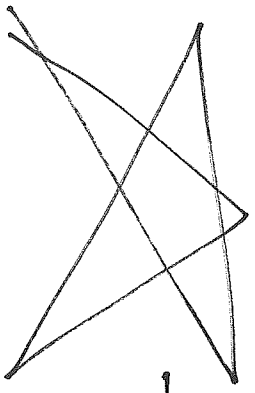


- Simple: Edges are non-intersecting
- convex: * All internal angles are less than 180°

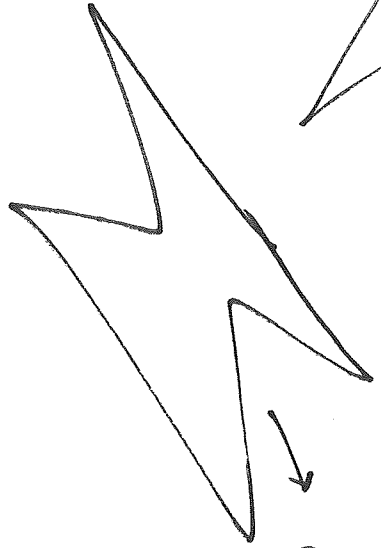
or equal to 180°
 * A diagonal will not intersect the edges

* All Δ s are convex!

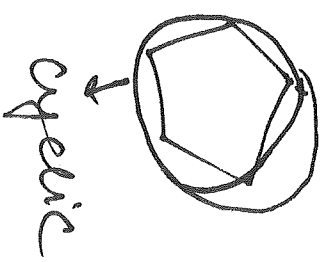
- nonconvex: special case of convex, always increasing in one direction



→ non simple



→ concave



↓ cyclic

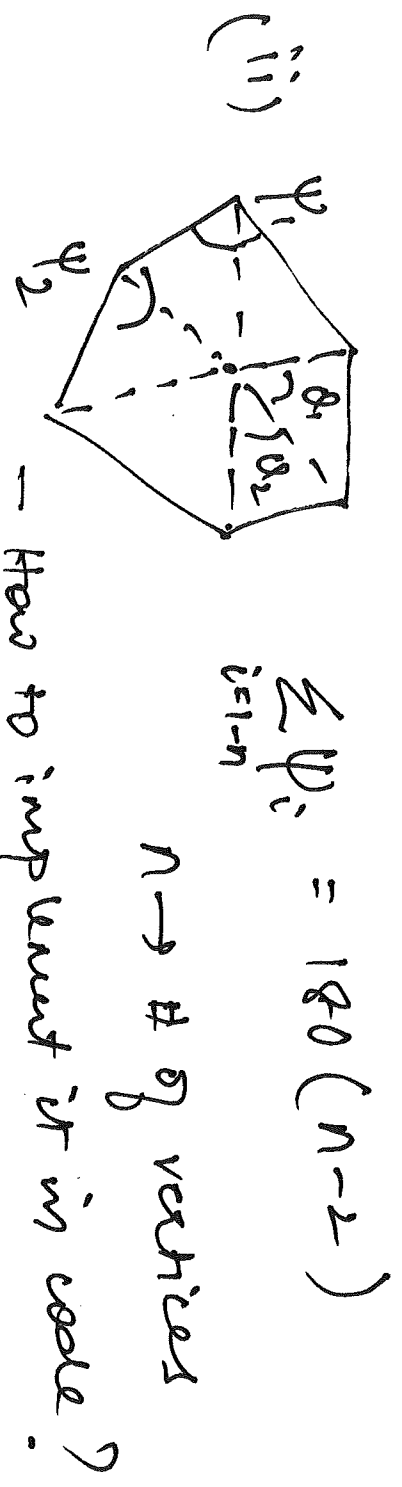
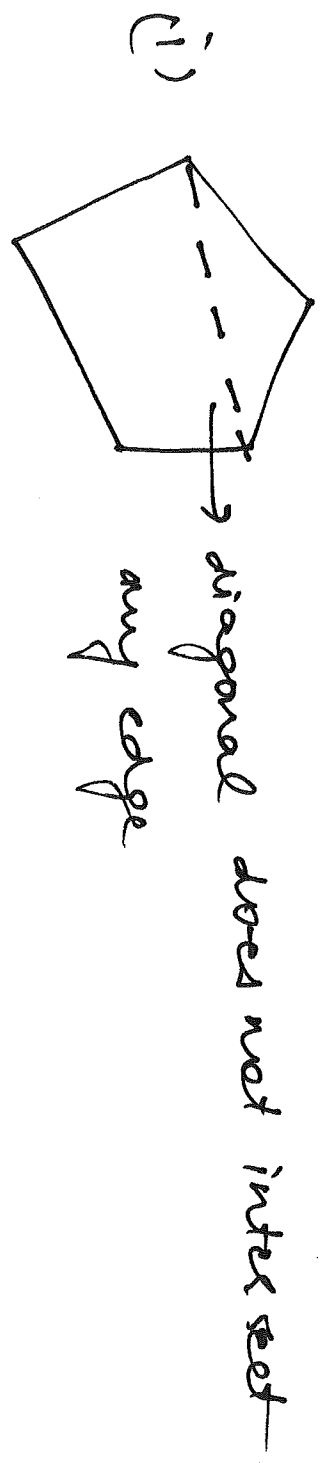


earni kataral

Polygons – ordering and normals

- Ordering of vertices
 - Counterclockwise – front facing, but can be changed using `glFrontFace()`
 - Hide a face using `glCullFace()`
- Front and back faces of a polygon
 - f.e. `glPolygonMode(GL_FRONT, GL_FILL)`
- Right hand and left hand rules
- Calculation of surface normal
 - How to determine the surface normal?
 - Why “normalize”?

Tests for convexity:-



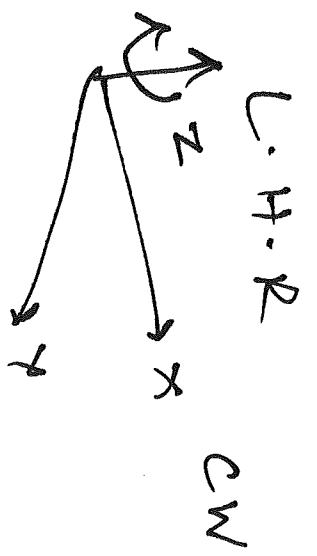
- First step
- Break each polygon to triangles
- Each triangle contributes 180°
- # of Δ 's * $180 =$ Sum of interior angles.

$$\sum_{i=1}^{n-1} \psi_i = 180(n-2)$$

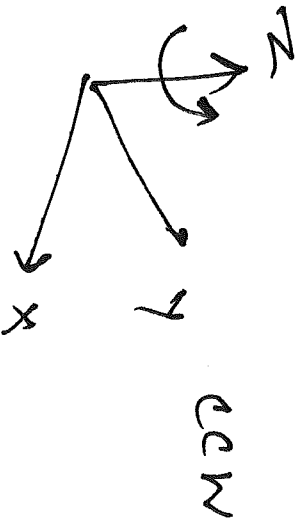
$n \rightarrow$ # of vertices

Vertex Ordering

can be clockwise



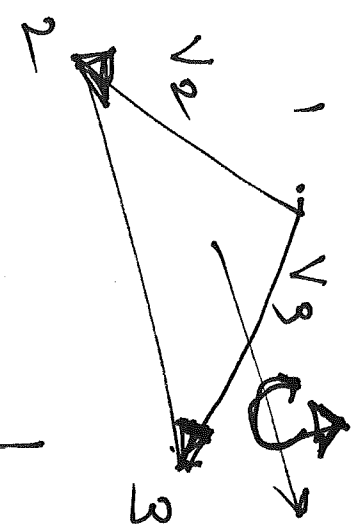
OR counter clockwise



- vertex ordering determines direction of normal
- OGL uses right handed coord sys

∴ we can ordering

$$\hat{n} = V_2 \times V_3$$

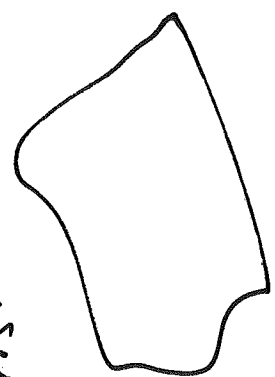


in matrix notation

	X	Y	Z
V _{2x}	V _{2y}	V _{2z}	
V _{3x}	V _{3y}	V _{3z}	

$$\frac{\vec{V}}{|\vec{V}|}$$

Why normalize?



- different objects, different scale & normal magnitude values
- OGL uses normals for lighting / shading

etc...
- Good idea to normalize!

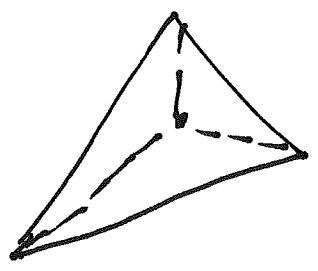
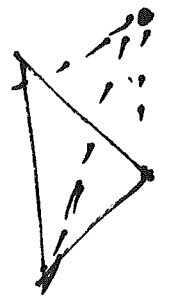
Polygons – intersection testing

- Point in Polygon
 - Given any point, determine if it is inside, lies on or is outside the polygon
 - Useful for ray tracing, collision detection and many other algorithms
 - Ray casting?
 - Winding number?
 - What is the sum of interior angles of a polygon?
 - $180*(n-2)$
 - Where will these algos fail? Where are they most suitable for?

Polygon, intersection testing:-

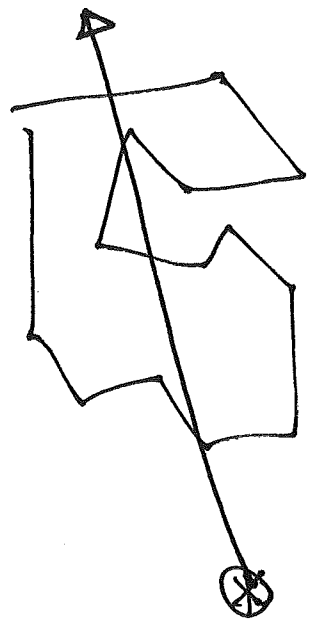
(4)

→ dot product, point in polygon.



$\sum_{i=1}^n \theta_i \approx 2\pi$
if point lies
in polygon.

→ Edge testing



- Pass ray from any pt in infinity
- If it intersects polygon even # times then it lies outside
- If it ~~lies~~ intersects odd # of edges, it lies inside!

Edge where edges can fail?

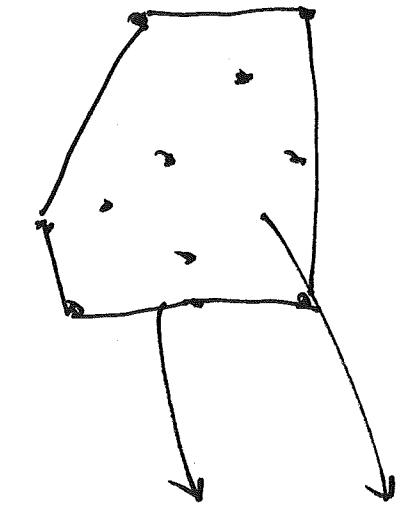
- ⊗ ray passes thro' vertex, point lies on vertex edge or polygon is non-coplanar.

Polygons – convex hulls

- Convex Hull
 - Polygon envelope
- For planar objects only
- The “gift wrapping” algorithm
 - Sort points
 - Determine point on extrema
 - Walk from extrema point in clockwise (or ccw) direction
 - Compare angles with all points in neighborhood with maximum subtended angle
- How can this be extended to polygons?

(5)

Given a number of points



The convex hull is the boundary enclosing those points

— It's also a polygon (in 2D)!

How to find the convex hull? (Gift wrapping algo)



— Take any point to be a chosen extrema (leftmost or rightmost)

— Find 'next' angle & w.r.t

clockwise points

— determine point with min

polar angle

— has to be at least 90° else point lies to its

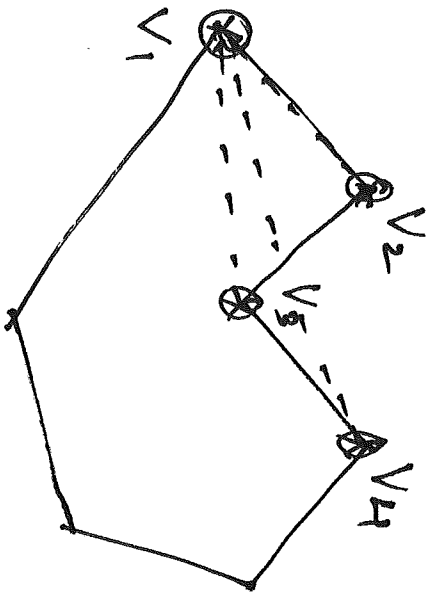
left

— Iterate.

— Same can be done through divide and conquer and other such methods.

⑥ - convex hull by definition is "convex"

- can we find convex hull of a "concave" polygon?



- choose extreme

(left or right)

- select next three consecutive vertices

- Angles from vertex

"0" to all 3 consecutive vertices must be in "progression")

{ increasing or decreasing }

- If any vertex fails the test, delete it and iterate through the other vertices.

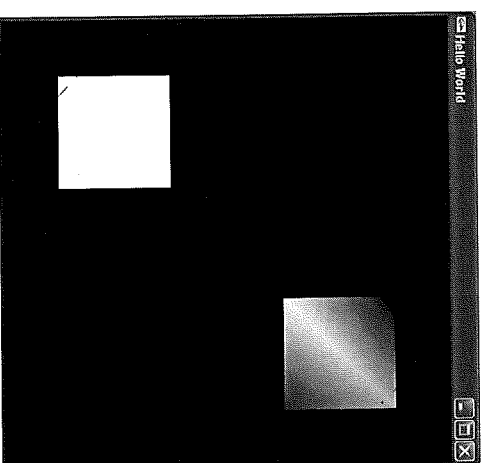
Coloring Your Objects

- Colors can either be from [0 – 1] or [0 – 255], we stick to former
- So typically all we need is glColor3f(R G B)
- What about Alpha ?
 - As long as we stick to just glColor(), alpha is not needed.
 - Will come into play when using materials, (later)
- Looking back at the polygon example
 - The first polygon is white in color, the second; each vertex has its own color
 - Color of the polygon is taken from the immediately preceding “state”
 - State Management
 - Number of things can affect the way primitives are rendered (graphical objects like lines, polygons etc.)
 - Things that affect them include color, lighting , texture etc. are called *states* or *state variables*
 - States can be turned “on” or “off” using glEnable and glDisable
 - Once a state is enabled it will affect all primitives succeeding it in code till it is disabled or another state is enabled which overrides the previous one.
 - F.e. color of polygon changing per object and per vertex.

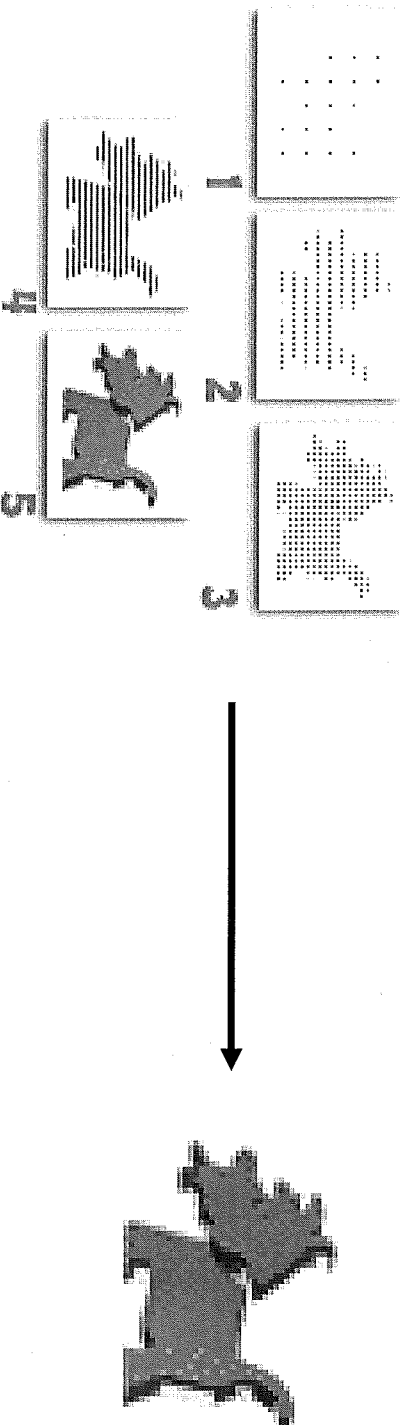
```
// Coloring
glColor3f( 1., 1., 1. );

// Matrix operation for rotation
glPushMatrix();
// For rotation
// glRotatef( spin, 0., 0., 1. );
// Actual code to do the drawing
glBegin( GL_POLYGON );
    glVertex3f( 0.25, 0.25, 0. );
    glVertex3f( 0.75, 0.25, 0. );
    glVertex3f( 0.75, 0.75, 0. );
    glVertex3f( 0.25, 0.75, 0. );
glEnd();
glPopMatrix();

//with colors
glBegin( GL_POLYGON );
    glColor3f( 1., 1., 1. );
    glVertex3f( 0.25, 0.25, 0. );
    glColor3f( 1., 0., 0. );
    glVertex3f( 0.75, 0.25, 0. );
    glColor3f( 0., 1., 0. );
    glVertex3f( 0.75, 0.75, 0. );
    glColor3f( 0., 0., 1. );
    glVertex3f( 0.25, 0.75, 0. );
glEnd();
```

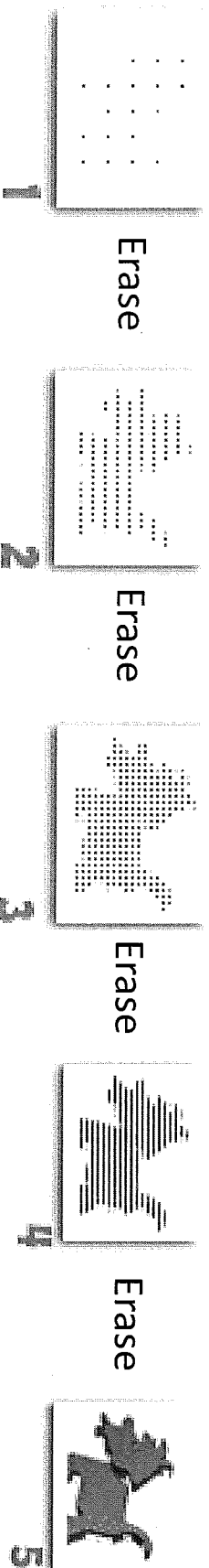


Basic Animation



Draw – Erase – Redraw , The way the computers do it !

Computationally more cheaper than updating changed pixels



Basic Animation

- Sequence of pictures at 24 frames per second
- Monitor refresh rate – 60 fps
- But also account for redraw and clearing times
- Double buffering, i.e. two frame buffers
 - One is drawn, the other displayed
 - Draw – Erase – Swap, repeat
- Difference between Single and Double Buffering
 - One artist with one sketchpad versus two artists with sketchpads working in tandem, one preparing the current drawing and the other preparing the next one.
- OpenGL does not inherently support double buffering
- Glut comes in here
 - glutSwapBuffers(), Forces the swapping of the two display buffers.
 - To be put at the end of the “display” code.
 - GLUT_DOUBLE, initializes the window to support double buffering