2IV60 Computer graphics
Graphics primitives and attributes

Jack van Wijk
TU/e

Overview

• Basic graphics primitives:
  – points, lines, polygons

• Basic attributes
  – linewidth, color

OpenGL output functions

```c
glBegin(GL_LINES);   // Specify what to draw,
// here lines
// Geometric info via vertices:
glVertex*(); // 1
...          // ...
glEnd;

[234][isfd]

[234]: 2D, 3D, 4D
[isfd]: integer, short, float, double
For instance: glVertex2i(100, 25);
```

Point and Line primitives

- **GL_POINTS**: sequence of points
- **GL_LINES**: sequence of line segments
- **GL_LINE_STRIP**: polyline
- **GL_LINE_LOOP**: closed polyline

Fill-area primitives 1

- Point, line and curve, fill area
- Usually polygons
- 2D shapes and boundary 3D objects

Fill-area primitives 2

- Approximation of curved surface:
  – Surface mesh or Surface tesselation
  – Face or facet (planar), patch (curved)
Polygons
• Polygon:
  – Planar shape, defined by a sequence of three or more vertices, connected by line segments (edges or sides)
  – Standard or simple polygon: no crossing edges

Regular polygons
• Vertices uniformly distributed over a circle:
  \[ (P_x, P_y) = (r \cos(i \theta / N), r \sin(i \theta / N)), i = 1 \ldots N \]

Convex vs. concave 1
• Convex:
  - All interior angles < 180 degrees, and
  - All line segments between 2 interior points in the polygon, and
  - All points at the same side of line through edge, and
  - From each interior point complete boundary visible

Convex vs. concave 2
• Puzzle:
  Given a planar polygon in 3D, with vertices \( P_i \), with \( i = 1, \ldots, N \).
  Give a recipe to determine if the polygon is convex or concave.

Solution: (multiple solutions possible)
- Assume concave, then:
  \[ Q_i = (P_i - P_{i+1}) \times (P_{i+1} - P_i) \]
  If \( Q_i > 0 \) for all \( i = 1, \ldots, N \), then convex, else concave.

Convex vs. concave 3
• Puzzle:
  Given a planar polygon in 3D, with vertices \( P_i \), with \( i = 1, \ldots, N \).
  Give a recipe to determine if the polygon is convex or concave.

Solution: (multiple solutions possible)
- Walk along vertices and determine if corner is convex or concave. Suppose, angle \( \angle P_{i-1} P_i P_{i+1} \) is concave.
  Split the polygon along the line through \( P_{i-1} \) and \( P_i \).
Convex polygon → triangles

Repeat
Pick three succeeding points;
Join the outside ones with a line;
Remove the middle point
Until three points are left over

Puzzle: Which sequence of points to pick?

Inside / outside polygon 1
Convex polygon. Check if a point $C$ is inside or outside a convex polygon in the $x,y$ plane.

Inside / outside polygon 2
Point $C$ is inside a convex polygon if all $Q_k$ have the same sign, with $Q_k = (P_i - C) \times (P_{i+1} - C)$.

Inside / outside polygon 3
• General: odd-even rule
– Draw a line from $C$ to a point far away. If the number of crossings with the boundary is odd, then $C$ is inside the polygon.

Storage polygons 1
• Mesh:
  – Vertices
  – Edges
  – Faces

Operations
– Draw all edges
– Draw all faces
– Move all vertices
– Determine orientation
– …

Storage polygons 2
• Simple: all polygons apart
Faces:
$F_1: V_1, V_2, V_3$  
$F_2: V_3, V_4, V_5$
Per vertex coordinates

H&B 4-7:84-94
Storage polygons 3

• More efficient: polygons and vertices apart

Faces: Vertices:
F\textsubscript{1}: 1,2,3
V\textsubscript{1}: x\textsubscript{1}, y\textsubscript{1}, z\textsubscript{1}

F\textsubscript{2}: 1,3,4,5
V\textsubscript{2}: x\textsubscript{2}, y\textsubscript{2}, z\textsubscript{2}

V\textsubscript{3}: x\textsubscript{3}, y\textsubscript{3}, z\textsubscript{3}
V\textsubscript{4}: x\textsubscript{4}, y\textsubscript{4}, z\textsubscript{4}
V\textsubscript{5}: x\textsubscript{5}, y\textsubscript{5}, z\textsubscript{5}
V\textsubscript{6}: x\textsubscript{6}, y\textsubscript{6}, z\textsubscript{6}

Storage polygons 4

• Also: polygons, edges and vertices apart

Faces: Edges: Vertices:
F\textsubscript{1}: 1,2,3
E\textsubscript{1}: 1,2
V\textsubscript{1}: x\textsubscript{1}, y\textsubscript{1}, z\textsubscript{1}

F\textsubscript{2}: 3,4,5,6
E\textsubscript{2}: 2,3
V\textsubscript{2}: x\textsubscript{2}, y\textsubscript{2}, z\textsubscript{2}

E\textsubscript{3}: 3,1
V\textsubscript{3}: x\textsubscript{3}, y\textsubscript{3}, z\textsubscript{3}

E\textsubscript{4}: 3,4
V\textsubscript{4}: x\textsubscript{4}, y\textsubscript{4}, z\textsubscript{4}

E\textsubscript{5}: 4,5
V\textsubscript{5}: x\textsubscript{5}, y\textsubscript{5}, z\textsubscript{5}

E\textsubscript{6}: 5,1
V\textsubscript{6}: x\textsubscript{6}, y\textsubscript{6}, z\textsubscript{6}

Storage polygons 5

• Or: keep list of neighboring faces per vertex

Faces: Vertices: Neighbor faces:
F\textsubscript{1}: 1,2,3
V\textsubscript{1}: x\textsubscript{1}, y\textsubscript{1}, z\textsubscript{1}
1, 2

F\textsubscript{2}: 1,3,4,5
V\textsubscript{2}: x\textsubscript{2}, y\textsubscript{2}, z\textsubscript{2}
1
1, 2
2

V\textsubscript{3}: x\textsubscript{3}, y\textsubscript{3}, z\textsubscript{3}
2

V\textsubscript{4}: x\textsubscript{4}, y\textsubscript{4}, z\textsubscript{4}

V\textsubscript{5}: x\textsubscript{5}, y\textsubscript{5}, z\textsubscript{5}

V\textsubscript{6}: x\textsubscript{6}, y\textsubscript{6}, z\textsubscript{6}

Storage polygons 6

• Many other formats possible
  – See f.i. winged edge data structure
• Select/design storage format based on:
  – Operations on elements;
  – Efficiency (size and operations);
  – Simplicity
  – Maintainability
  – Consistency
  – …

Polygons in space 1

• 3D flat polygon in (infinite) plane
• Equation plane:
  \[ Ax + By + Cz + D = 0 \]

Plane: z=0

Polygons in space 2

• Position point (x, y, z) w.r.t. plane:
  \[ Ax + By + Cz + D > 0 \]: point in front of plane
  \[ Ax + By + Cz + D < 0 \]: point behind plane

Plane: z=0
Polylons in space 3

- Normal: vector $N$ perpendicular to plane
- Unit normal: $|N|=1$

Normal: $(0, 0, 1)$

Plane: $Ax + By + Cz + D = 0$

In general:
- Normal: $N=(A, B, C)$ for $N \cdot X + D = 0$
- Normalized: $N$ is a unit vector

OpenGL Fill-Area Functions 1

```c
glBegin(GL_POLYGON);   // Specify what to draw, // a polygon
// Geometric info via vertices:
glVertex*(); // 1
glVertex*(); // 2
...     // ...
glEnd;

[234][isfd]
[isfd]: integer, short, float, double
For instance: glVertex2f(100, 25);
```

OpenGL Fill-Area Functions 2

```c
GL_POLYGON: convex polygon

Concave polygons give unpredictable results.
```

OpenGL Fill-Area Functions 3

- $\text{GL}_\text{TRIANGLES}$: sequence of triangles
- $\text{GL}_\text{TRIANGLE}\_\text{STRIP}$:
- $\text{GL}_\text{TRIANGLE}\_\text{FAN}$:

OpenGL Fill-Area Functions 4

- $\text{GL}_\text{QUADS}$: sequence of quadrilaterals
- $\text{GL}_\text{QUAD}\_\text{STRIP}$: strip of quadrilaterals
More efficiency

• Reduce the number of function calls:
  – OpenGL Vertex Arrays: pass arrays of vertices instead of single ones;

• Store information on GPU and do not resend:
  – OpenGL Display lists;
  – Vertex Buffer Objects.

OpenGL Display lists 1

Key idea: Collect a sequence of OpenGL commands in a structure, stored at the GPU, and reference it later on
+ Simple to program
+ Can give a boost
+ Useful for static scenes and hierarchical models
– Not the most modern

OpenGL Display lists 2

// Straightforward

void drawRobot();
{ // lots of glBegin, glVertex, glEnd calls }

void drawScene(); {
  drawRobot();
  glTranslate3f(1.0, 0.0, 0.0);
  drawRobot();
}

OpenGL Display lists 3

void drawRobot();
{ // lots of glBegin, glVertex, glEnd calls }
int rl_id;

void init() { 
  rl_id = glGenLists(1);         // get id for list
  glNewList(rl_id, GL_COMPILE);  // create new list
drawing commands;             // draw something
glEndList(); }                 // end of list

void drawScene(); {
  glCallList(rl_id);             // draw list
  glTranslate3f(1.0, 0.0, 0.0);
  glCallList(rl_id);             // and again
}

OpenGL Display lists 4

First, get an id. Either some fixed constant, or get a guaranteed unique one:

rl_id = glGenLists(1);         // get id for list

Next, create a display list with this id:

glNewList(rl_id, GL_COMPILE);  // create new list
drawing commands;             // draw something
glEndList(); }                 // end of list

Finally, “replay” the list. Change the list only when the scene is changed:

glCallList(rl_id);             // draw list

Attributes 1

• Attribute parameter: determines how a graphics primitive is displayed
• For instance for line: color, width, style
• OpenGL: simple state system
  – Current setting is maintained;
  – Setting can be changed with a function.
Attributes 2

Example:
```c
.glLineWidth(width);
.glEnable(GL_LINE_STIPPLE);
.glLineStipple(repeatfactor, pattern);
// draw stippled lines
...;
.glDisable(GL_LINE_STIPPLE);
```

Note: special features often must be enabled explicitly with `glEnable()`.

H&B 5-7:141-143

Color 1

RGB mode:
- color = (red, green, blue)
- `glColor[34][bisf]`: set color of vertex
- For example:
  - `glColor4f(red, green, blue, alpha)`
  - `alpha`: opacity (1–transparent)

H&B 5-2:130-131

Color 2

- RGB: hardware oriented model
- Dimensions are not natural
- (0, 0, 0): black; (1, 0, 0): red; (1, 1, 1) white

H&B 5-2:130-131

Color 3

- HSV: Hue, Saturation (Chroma), Value
- More natural dimensions
  - Hue: red-orange-yellow-green-blue-purple;
  - Saturation: grey-pink-red;
  - Value: dark-light

H&B 19-7:614-617

Color 4

- HSL: Hue, Saturation (Chroma), Lightness
- More natural dimensions

H&B 19-8:618-618

Color 5

- Color: big and complex topic
- Color: not physical, but perceptual

H&B 5-2:130-131
Next

- We now know how to draw polygons.
- How to model geometric objects?