# 2IV60 Computer graphics
# Graphics primitives and attributes

Jack van Wijk
TU/e

## Overview

- Basic graphics primitives:
  - points, lines, polygons
- Basic attributes
  - linewidth, color

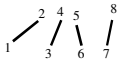
## OpenGL output functions

```
glBegin(GL_LINES);    // Specify what to draw,
                      // here lines

// Geometric info via vertices:
   glVertex*(); // 1
   glVertex*(); // 2
   ...          // ...
glEnd;

glVertex[234][isfd]

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

H&B 4-4:81-82

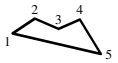
## Point and Line primitives


**GL_POINTS**: sequence of points




**GL_LINES**: sequence of line segments


**GL_LINE_STRIP**: polyline




**GL_LINE_LOOP**: closed polyline

H&B 4-4:81-82

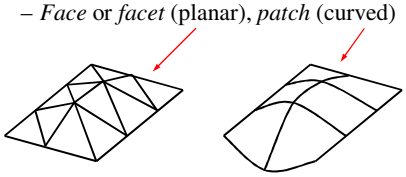
## Fill-area primitives 1

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

H&B 4-6:83-84

## Fill-area primitives 2

- Approximation of curved surface:
  - *Surface mesh* or *Surface tesselation*
  - *Face* or *facet* (planar), *patch* (curved)



H&B 4-6:83-84

# Polygon

- 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

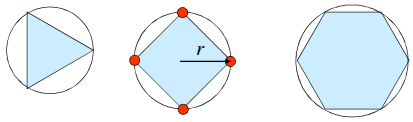
*bowtie polygon*

H&B 4-7:84-94

# Regular polygon

- Vertices uniformly distributed over a circle:

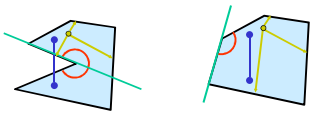
$$(\mathbf{P}x_i, \mathbf{P}y_i) = (r\cos(i2\pi/N), r\sin(i2\pi/N)), i = 1...N$$




H&B 4-7:84-94

# Convex vs. concave 1

- *Convex*:
  - All interior angles < 180 graden, and
  - All line segments between 2 interior points in polygon, and
  - All points at the same side of line through edge, and
  - From each interior point complete boundary visible
- *Concave*: not convex



H&B 4-7:84-94

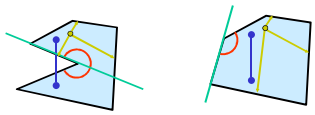
# Convex vs. concave 2

- *Puzzle:*

  Given a planar polygon in 3D,

  with vertices $\mathbf{P}_i$, with i = 1, … , *N*.

  Give a recipe to determine if the polygon is convex or concave.



H&B 4-7:84-94

# Convex vs. concave 3

- *Puzzle:*

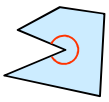
  Given a planar polygon in 3D,

  with vertices $\mathbf{P}_i$, with i = 1, … , *N*.

  Give a recipe to determine if the polygon is convex or concave.

  *Solution: (multiple solutions possible)*

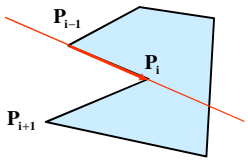
  If $\mathbf{Q}_{i+1} \cdot \mathbf{Q}_i > 0$ for all $i = 1,...,N$, with $\mathbf{Q}_i = (\mathbf{P}_i - \mathbf{P}_{i-1}) \times (\mathbf{P}_{i+1} - \mathbf{P}_i)$, then convex, else concave. Assume here that $\mathbf{P}_{i+N} = \mathbf{P}_i$.



H&B 4-7:84-94

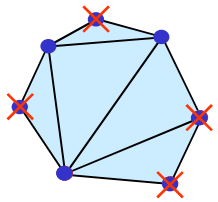
# Splitting concave polygons

Walk along vertices and determine if corner is convex or concave. Suppose, angle $\mathbf{P}_{i-1}\mathbf{P}_i\mathbf{P}_{i+1}$ is concave. Split the polygon along the line through $\mathbf{P}_{i-1}$ en $\mathbf{P}_i$.




H&B 4-7:84-94

## Convex polygon → triangles



Repeat
- Pick three succeeding points;
- Join the outside ones with a line;
- Remove the middle point

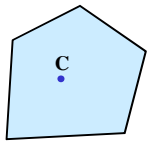
Until three points are left over

Puzzle: Which sequence of points to pick?

H&B 4-7:84-94

## Inside / outside polygon 1

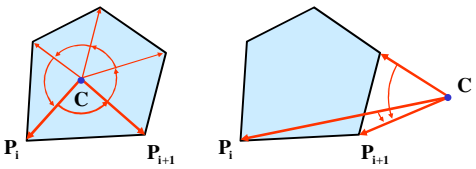
Convex polygon.
Check if a point **C** is inside or outside a convex polygon in the $x$, $y$ plane.




H&B 4-7:84-94

## Inside / outside polygon 2

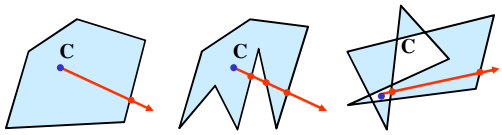
Point **C** is inside a convex polygon if all $Q_{iz}$ have the same sign, with $\mathbf{Q_i} = (\mathbf{P_i} - \mathbf{C}) \times (\mathbf{P_{i+1}} - \mathbf{C})$.




H&B 4-7:84-94

## 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 polygoon.

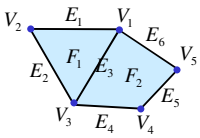



H&B 4-7:84-94

## Storage polygons 1

- Mesh:
  - Vertices
  - Edges
  - Faces

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



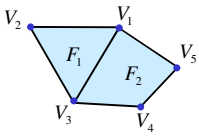

H&B 4-7:84-94

## Storage polygons 2

- Simple: all polygons apart

Faces:
$F_1$: $V_1$, $V_2$, $V_3$
$F_2$: $V_1$, $V_3$, $V_4$, $V_5$

Per vertex coordinates



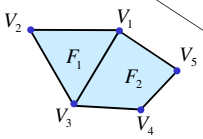

H&B 4-7:84-94

## Storage polygons 3

- More efficient: polygons and vertices apart

Faces:
$F_1$: 1,2,3
$F_2$: 1,3,4,5

Vertices:
$V_1$: $x_1$, $y_1$, $z_1$
$V_2$: $x_2$, $y_2$, $z_2$
$V_3$: $x_3$, $y_3$, $z_3$
$V_4$: $x_4$, $y_4$, $z_4$
$V_5$: $x_5$, $y_5$, $z_5$
$V_6$: $x_6$, $y_6$, $z_6$




H&B 4-7:84-94

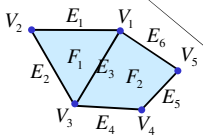
## Storage polygons 4

- Also: polygons, edges and vertices apart

Faces:
$F_1$: 1,2,3
$F_2$: 3,4,5,6

Edges:
$E_1$: 1,2
$E_2$: 2,3
$E_3$: 3,1
$E_4$: 3,4
$E_5$: 4,5
$E_6$: 5,1

Vertices:
$V_1$: $x_1$, $y_1$, $z_1$
$V_2$: $x_2$, $y_2$, $z_2$
$V_3$: $x_3$, $y_3$, $z_3$
$V_4$: $x_4$, $y_4$, $z_4$
$V_5$: $x_5$, $y_5$, $z_5$
$V_6$: $x_6$, $y_6$, $z_6$



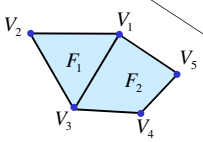

H&B 4-7:84-94

## Storage polygons 5

- Or: keep list of neighboring faces per vertex

Faces:
$F_1$: 1,2,3
$F_2$: 1,3,4,5

| Vertices: | Neighbor faces: |
|---|---|
| $V_1$: $x_1$, $y_1$, $z_1$ | 1, 2 |
| $V_2$: $x_2$, $y_2$, $z_2$ | 1 |
| $V_3$: $x_3$, $y_3$, $z_3$ | 1, 2 |
| $V_4$: $x_4$, $y_4$, $z_4$ | 2 |
| $V_5$: $x_5$, $y_5$, $z_5$ | 2 |
| $V_6$: $x_6$, $y_6$, $z_6$ | … |




H&B 4-7:84-94

## Storage polygons 6

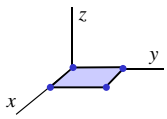
- Many other formats possible
  - See f.i. *winged edge data structure*
- Select/design storage format based on:
  - Operations on elements;
  - Efficiency (size and operations);
  - Simplicity
  - Maintainability
  - Consistency
  - …

H&B 4-7:84-94

## Polygons in space 1

- 3D flat polygon in (infinite) plane
- Equation plane:

$Ax + By + Cz + D = 0$



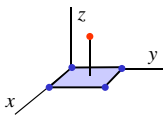

Plane: $z$=0

H&B 4-7:84-94

## 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

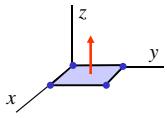



Plane: $z$=0

H&B 4-7:84-94

## Polygons in space 3

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




Normal: (0, 0, 1)
Vlak: $z$=0

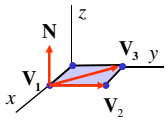
In general:
Normal: **N**=($A$, $B$, $C$) for
Plane: $Ax + By + Cz + D = 0$
Or **N.X**+$D$ = 0

H&B 4-7:84-94

## Polygons in space 4

Given vertices of a polygon,
determine plane equation $Ax + By + Cy + D = 0$.

Take three arbitrary vertices $\mathbf{V_1}, \mathbf{V_2}$ en $\mathbf{V_3}$.
Calculate the normal vector:
$\mathbf{N} = (\mathbf{V_2} - \mathbf{V_1}) \times (\mathbf{V_3} - \mathbf{V_1})$.
$\mathbf{V_1}$ is in the plane, hence $\mathbf{N} \cdot \mathbf{V_1} + D = 0$.
In short:
$A = \mathbf{N}_x, B = \mathbf{N}_y, C = \mathbf{N}_z, D = -\mathbf{N} \cdot \mathbf{V_1}$.




H&B 4-7:84-94

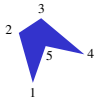
## OpenGL Fill-Area Functions 1

```
glBegin(GL_POLYGON);    // Specify what to draw,
                        // a polygon

// Geometric info via vertices:
   glVertex*(); // 1
   glVertex*(); // 2
   ...          // ...
glEnd;

glVertex[234][isfd]

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

H&B 4-4:81-82

## OpenGL Fill-Area Functions 2


**GL_POLYGON**: *convex* polygon

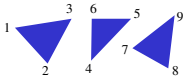



*Concave* polygons give unpredictable results.

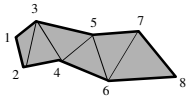
H&B 4-8:94-99

## OpenGL Fill-Area Functions 3

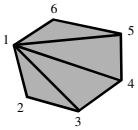
- **GL_TRIANGLES**: sequence of triangles




- **GL_TRIANGLE_STRIP**:



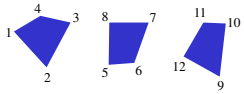

- **GL_TRIANGLE_FAN**:



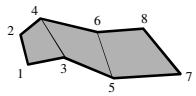

H&B 4-8:94-99

## OpenGL Fill-Area Functions 4

- **GL_QUADS**: sequence of quadrilaterals




- **GL_QUAD_STRIP**: strip of quadrilaterals




H&B 4-8:94-99

## 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.

H&B App. C

## 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

H&B 4-15 111-113

## OpenGL Display lists 2

```
// Straightforward

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

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

H&B 4-15 111-113

## 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
   drawRobot();                    // draw your object
   glEndList(); }                  // end of list

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

H&B 4-15 111-113

## 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
```

H&B 4-15 111-113

## 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.

H&B 5-1:130

## Attributes 2

Example:

```
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

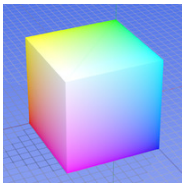
RGB mode :

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

H&B 5-2:130-131

## Color 2

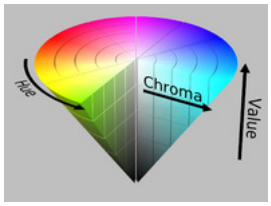
- RGB: hardware oriented model
- Dimensions are not natural
- (0, 0, 0): black; (1, 0, 0): red; (1, 1, 1) white



H&B 5-2:130-131

## Color 3

- HSV: Hue, Saturation (Chroma), Value
- More natural dimensions

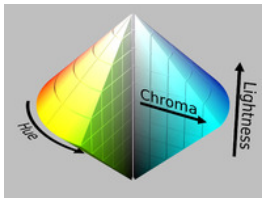


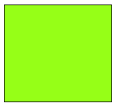

- Hue: red-orange-yellow-green-blue- purple;
- Saturation: grey-pink-red;
- Value: dark-light

H&B 19-7:614-617

## Color 4

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

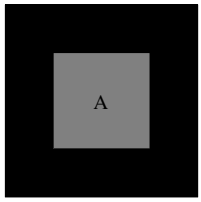


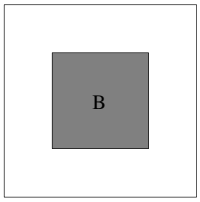



H&B 19-8:618-618

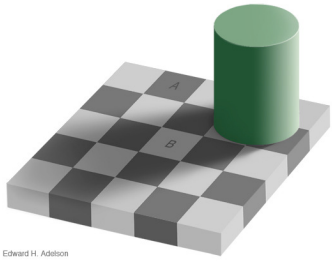
## Color 5

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



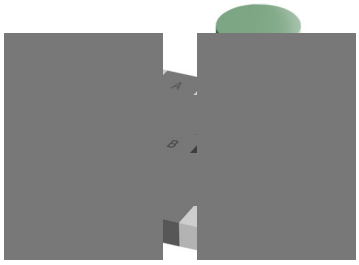



## Color 6




- Adelson checkerboard illusion

## Color 6




- Adelson checkerboard illusion

## Next

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