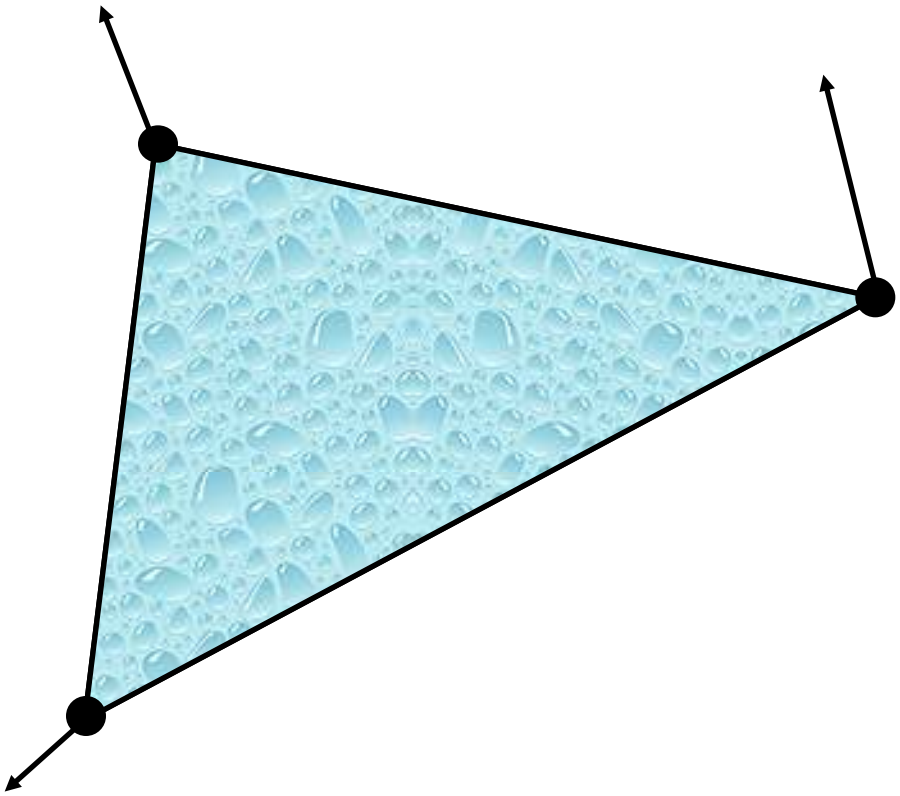
Overview

- Introduction to 3D graphics
- Matrices and algebra
- GPU programming

Overview

- *Introduction to 3D graphics*
- Matrices and algebra
- GPU programming

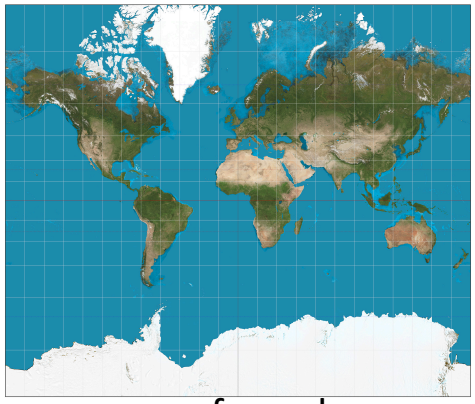
Triangles



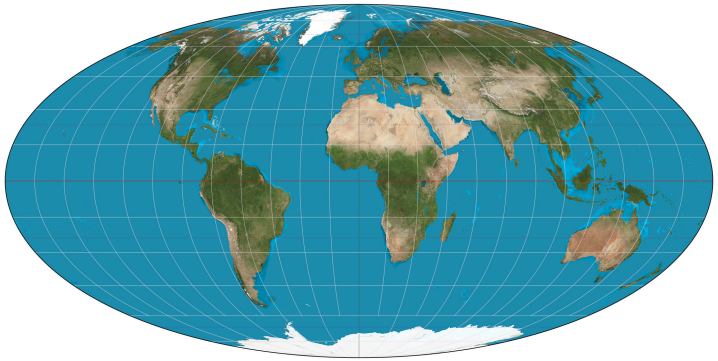
```
glTexCoord2f(tu1,tv1)
glNormal3f(nx1,ny1,nz1)
glVertex3f(x1,y1,z1)

glTexCoord2f(tu2,tv2)
glNormal3f(nx2,ny2,nz2)
glVertex3f(x2,y2,z2)

glTexCoord2f(tu3,tv3)
glNormal3f(nx3,ny3,nz3)
glVertex3f(x3,y3,z3)
```

Surface parameterization

Find a bijective mapping between a given surface and 2D parameter domain

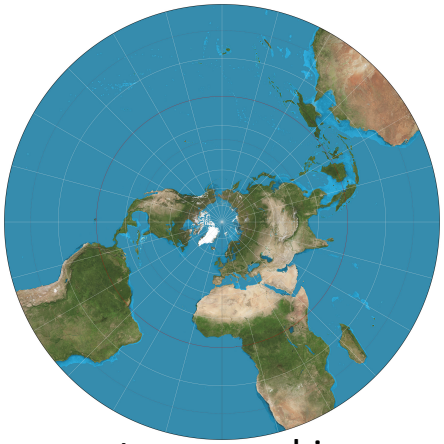


conformal



equiareal

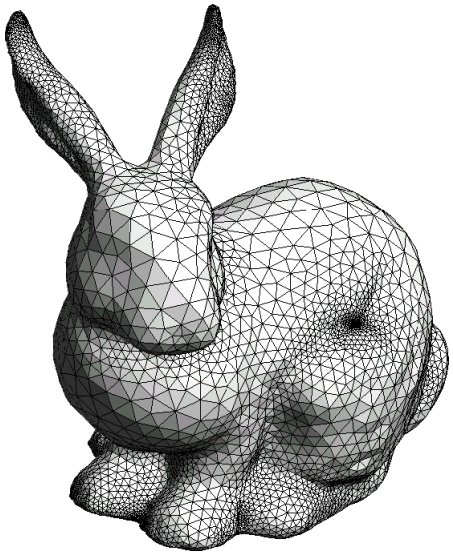
orthographic

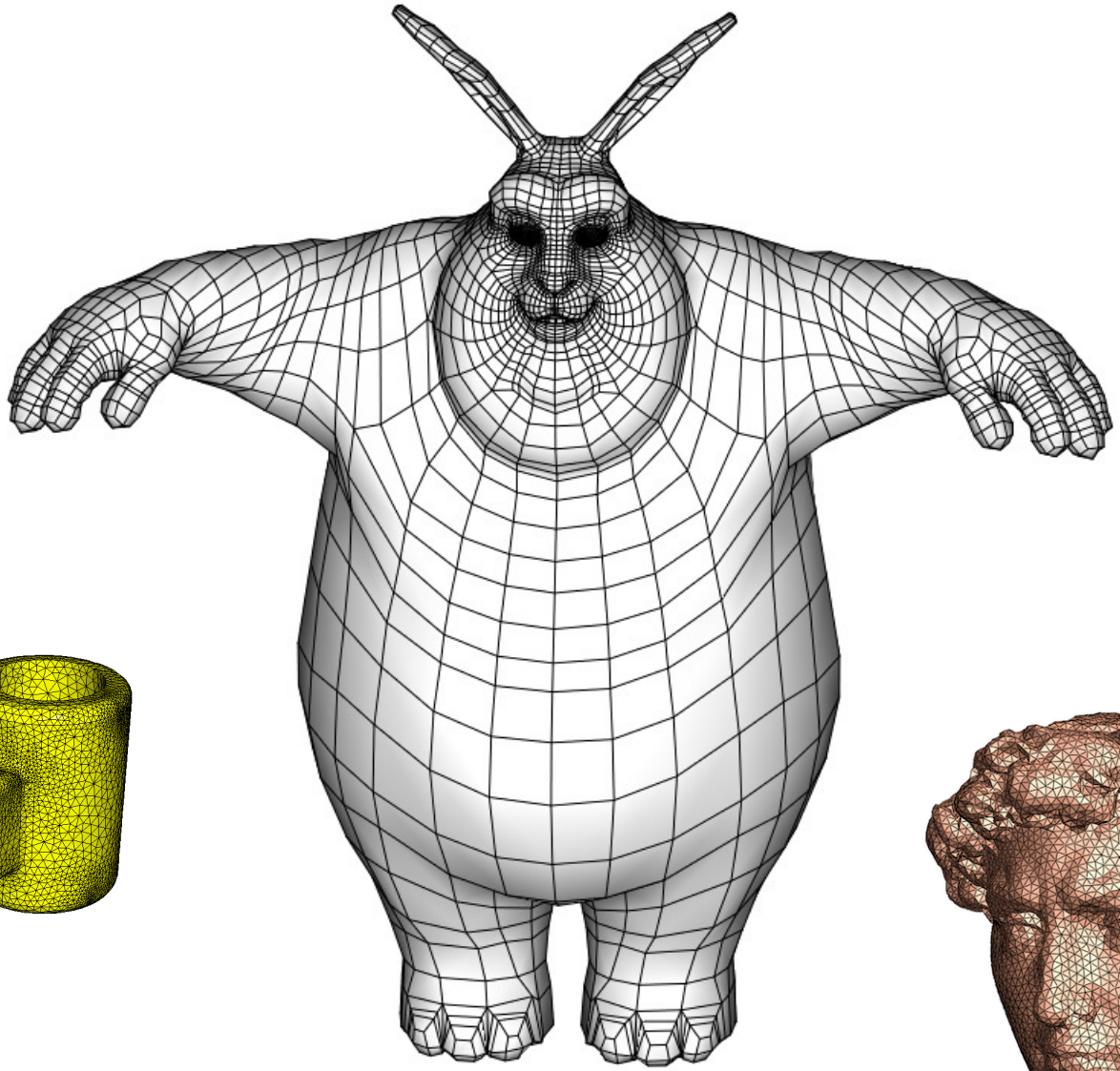


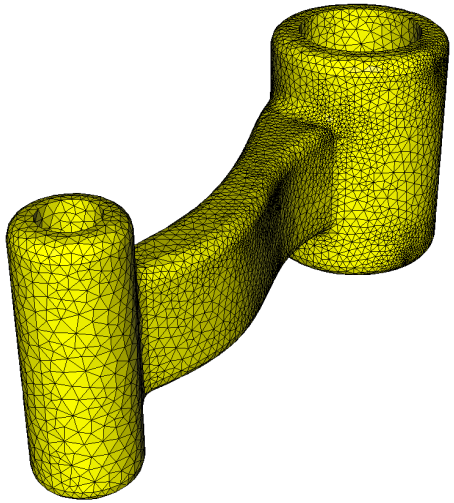
stereographic

...

Meshes

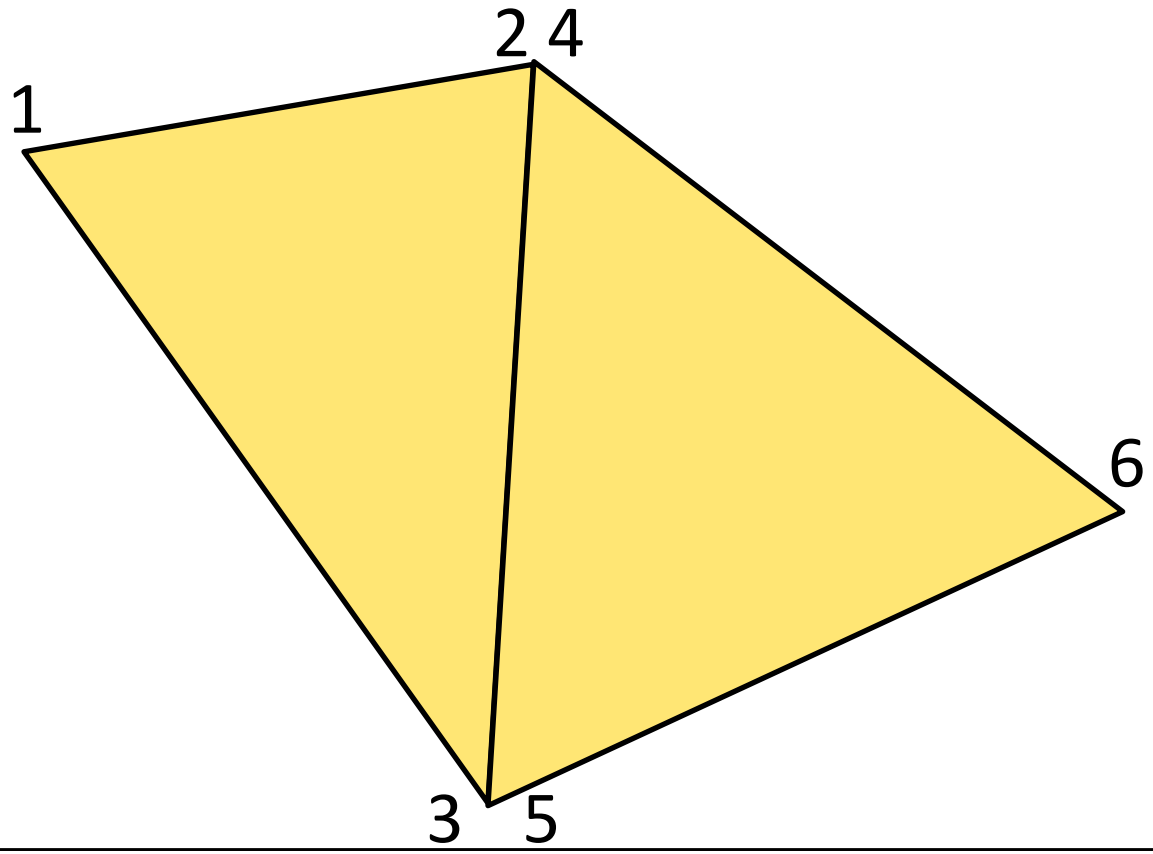






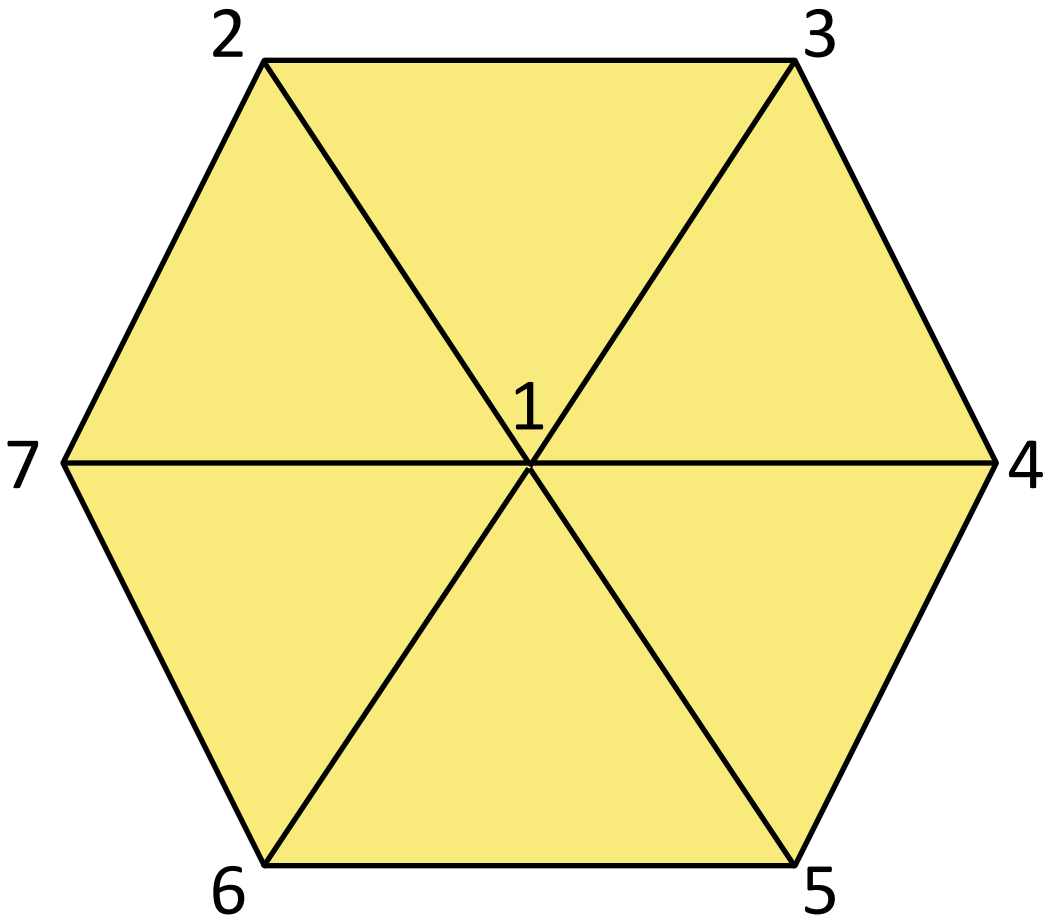
Triangles

```
glBegin(GL_TRIANGLES)
glVertex3f(v1)
glVertex3f(v2)
…
glVertex3f(v6)
glEnd()
```



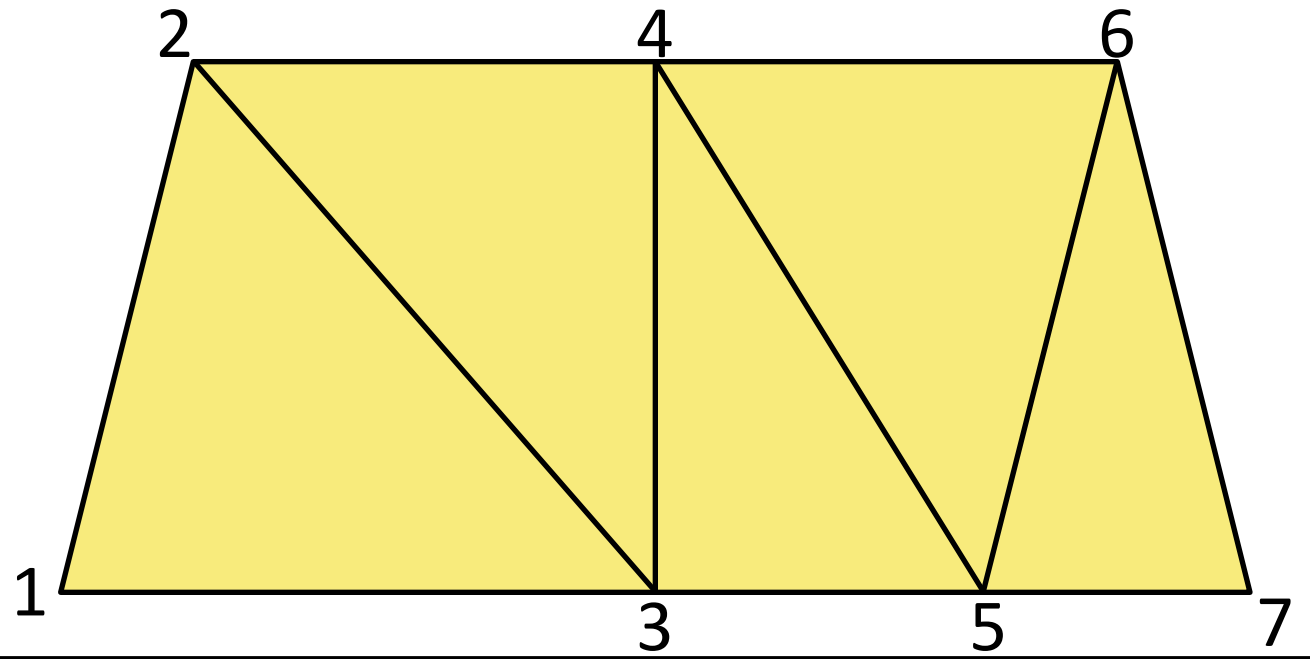
Triangle fan

```
glBegin(GL_TRIANGLE_FAN)
glVertex3f(v1)
glVertex3f(v2)
…
glVertex3f(v7)
glEnd()
```

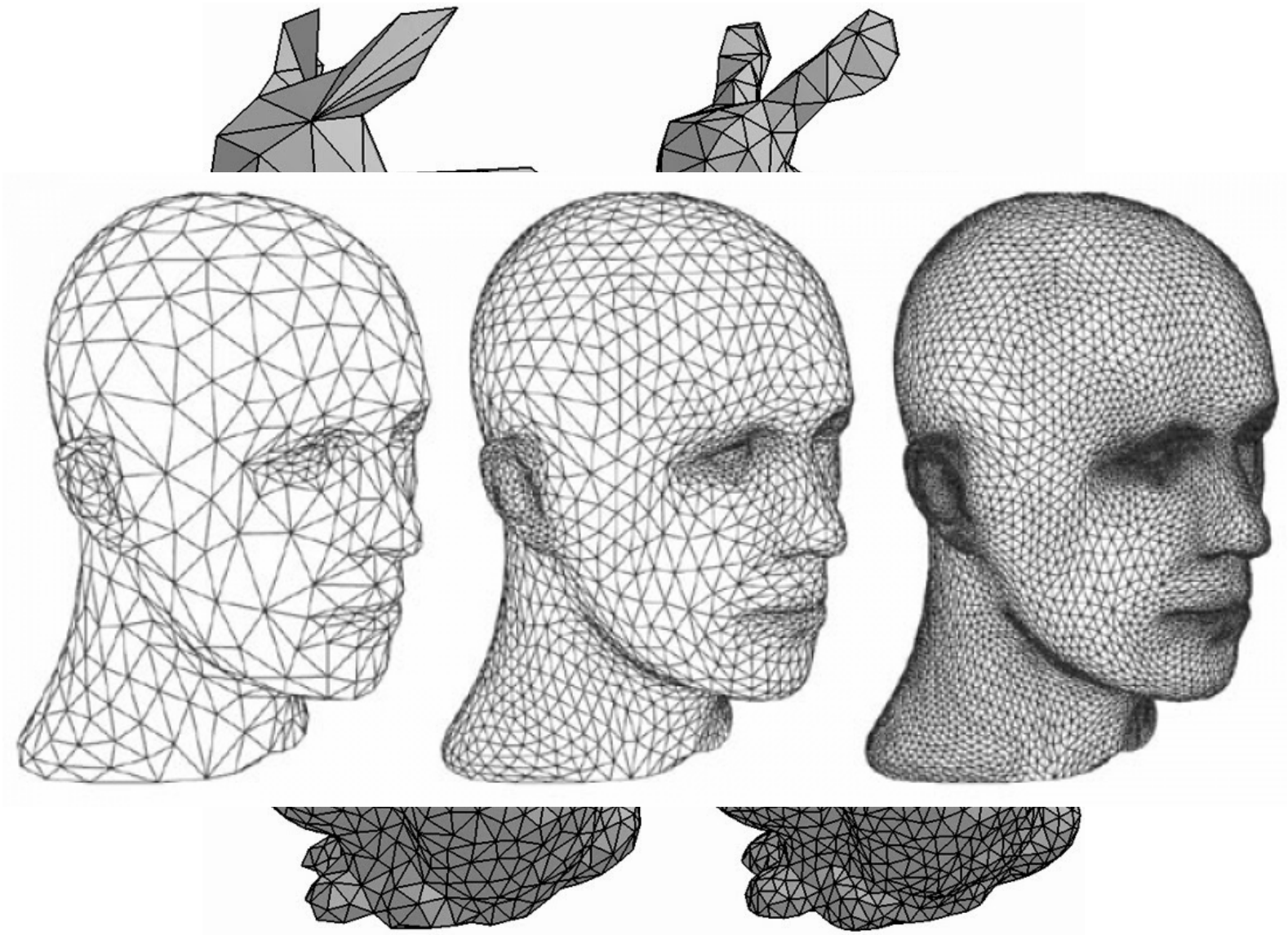


Triangle strip

```
glBegin(GL_TRIANGLE_STRIP)
glVertex3f(v1)
glVertex3f(v2)
…
glVertex3f(v7)
glEnd()
```



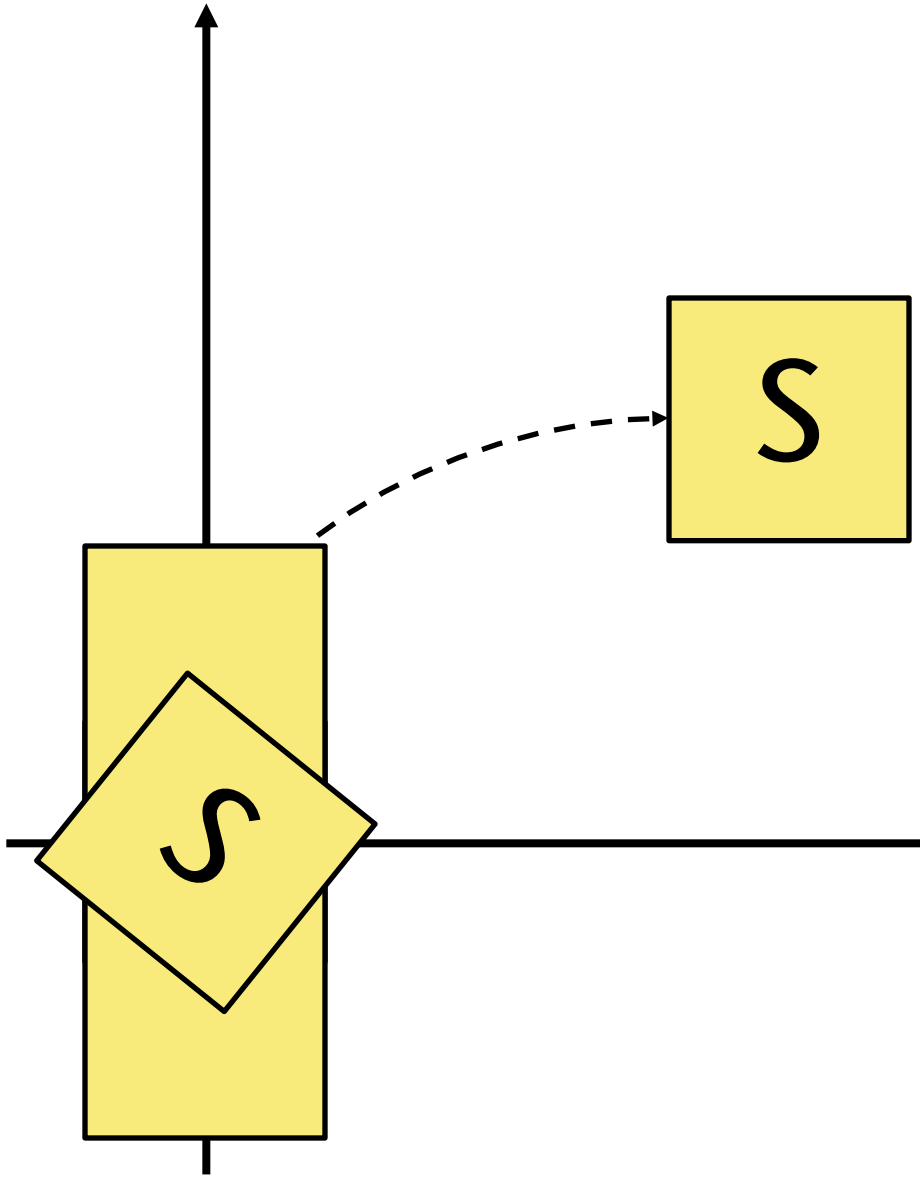
Resolution



Overview

- Introduction to 3D graphics
- *Matrices and algebra*
- GPU programming

Transformations



- Translation
- Scaling
- Rotation

Transformations

- Translation $$p' = p + t = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

- Scaling $$p' = D \cdot p = \begin{pmatrix} d_x & 0 & 0 \\ 0 & d_y & 0 \\ 0 & 0 & d_z \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

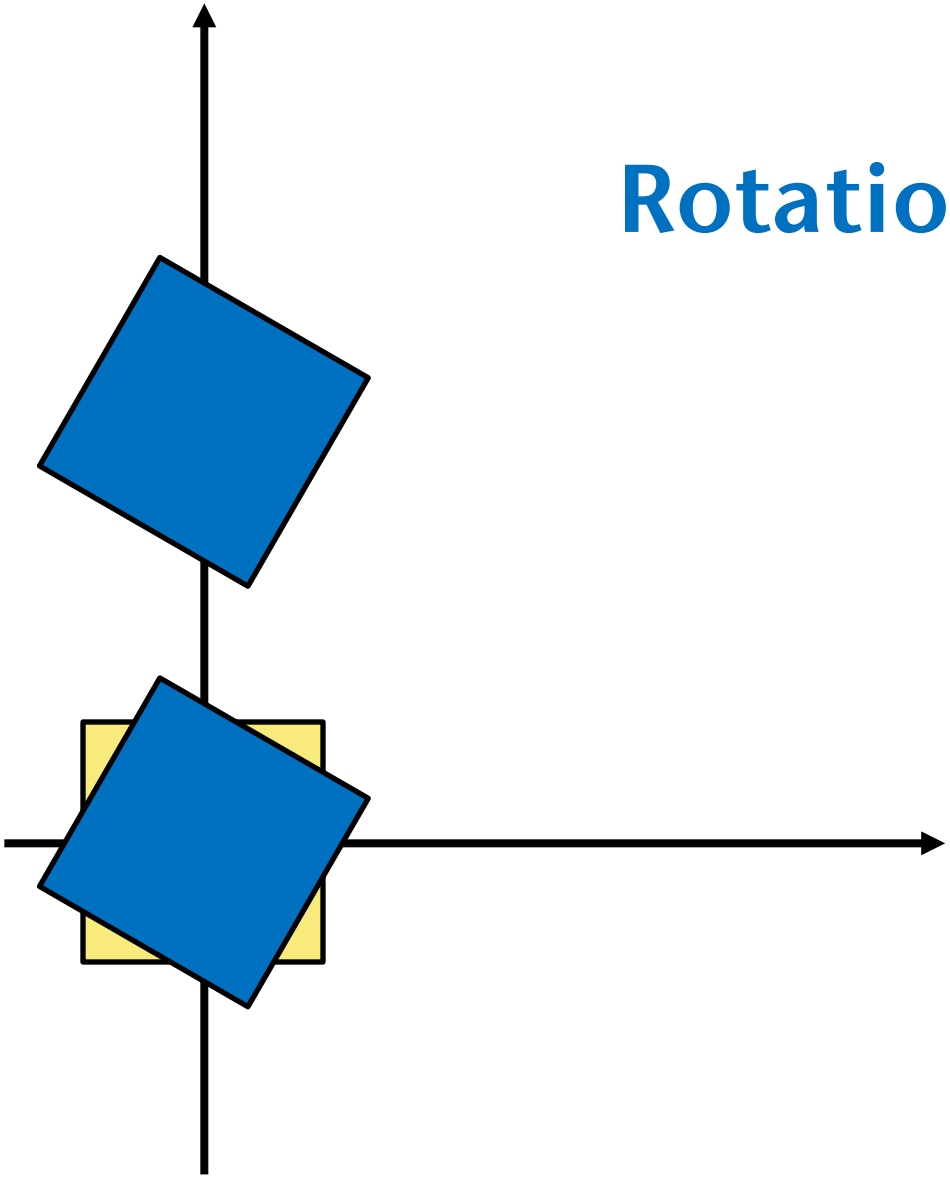
- Rotation $p' = R_\alpha^x \cdot p = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Homogenous coordinates

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad \tilde{p} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \qquad \tilde{t} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

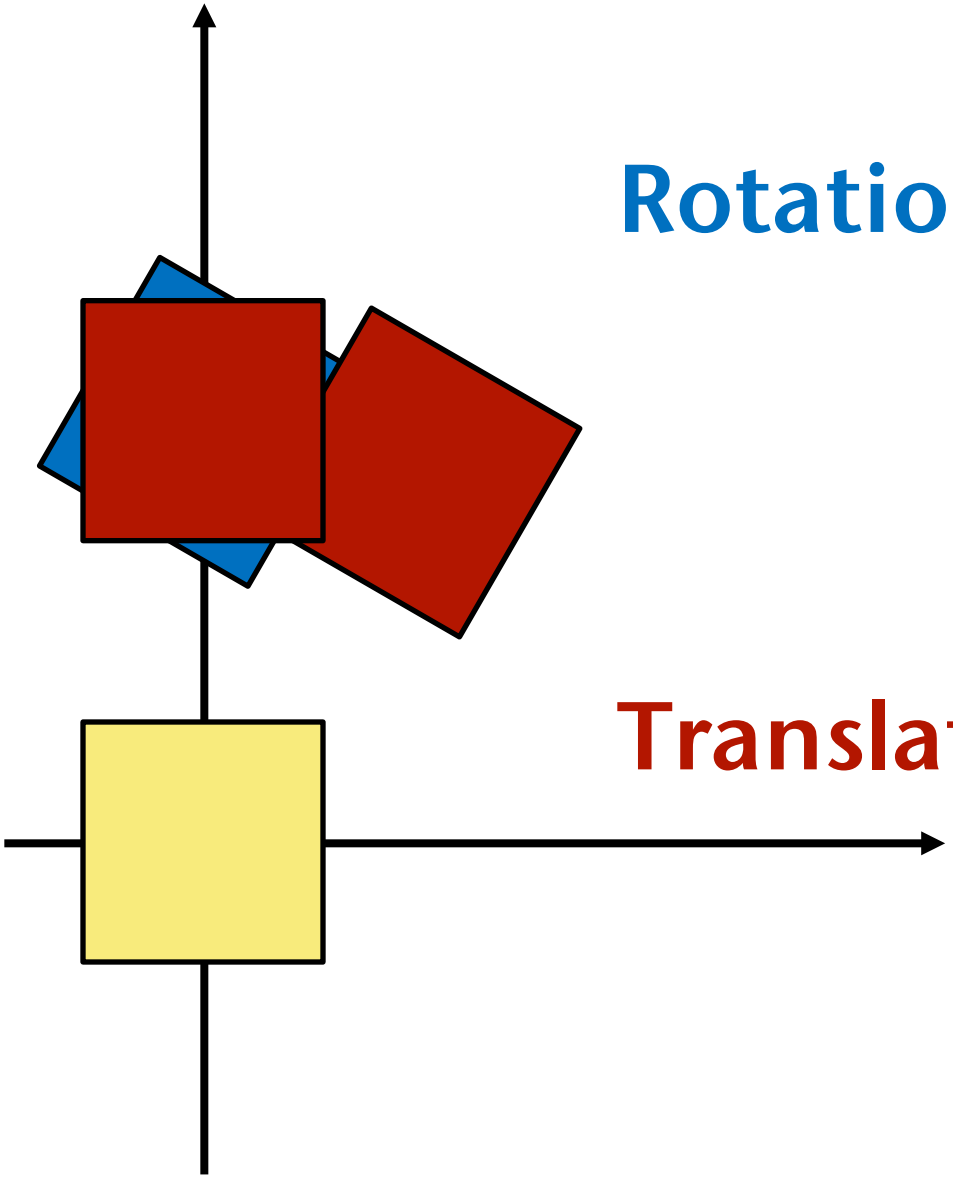
$$\tilde{D} = \begin{pmatrix} d_x & 0 & 0 & 0 \\ 0 & d_y & 0 & 0 \\ 0 & 0 & d_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \tilde{R}^x_\alpha = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformation order



Rotation THEN translation

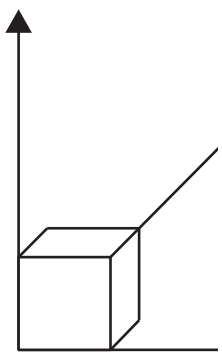
Transformation order

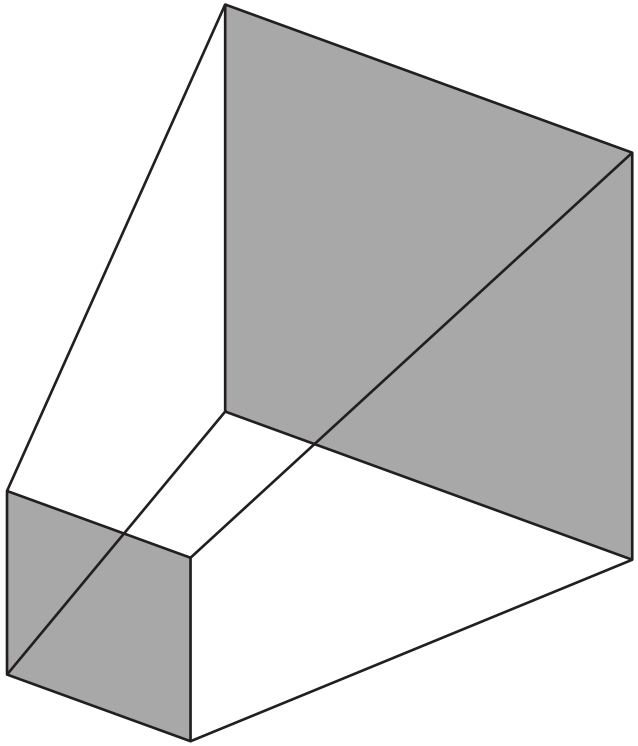


Rotation THEN translation

Translation THEN Rotation

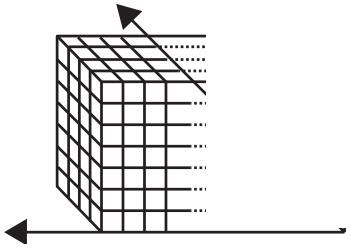
Coordinates pipeline

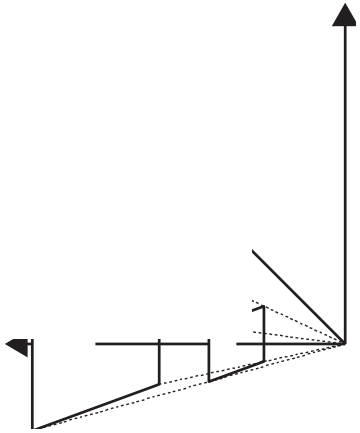




far plane

near plane



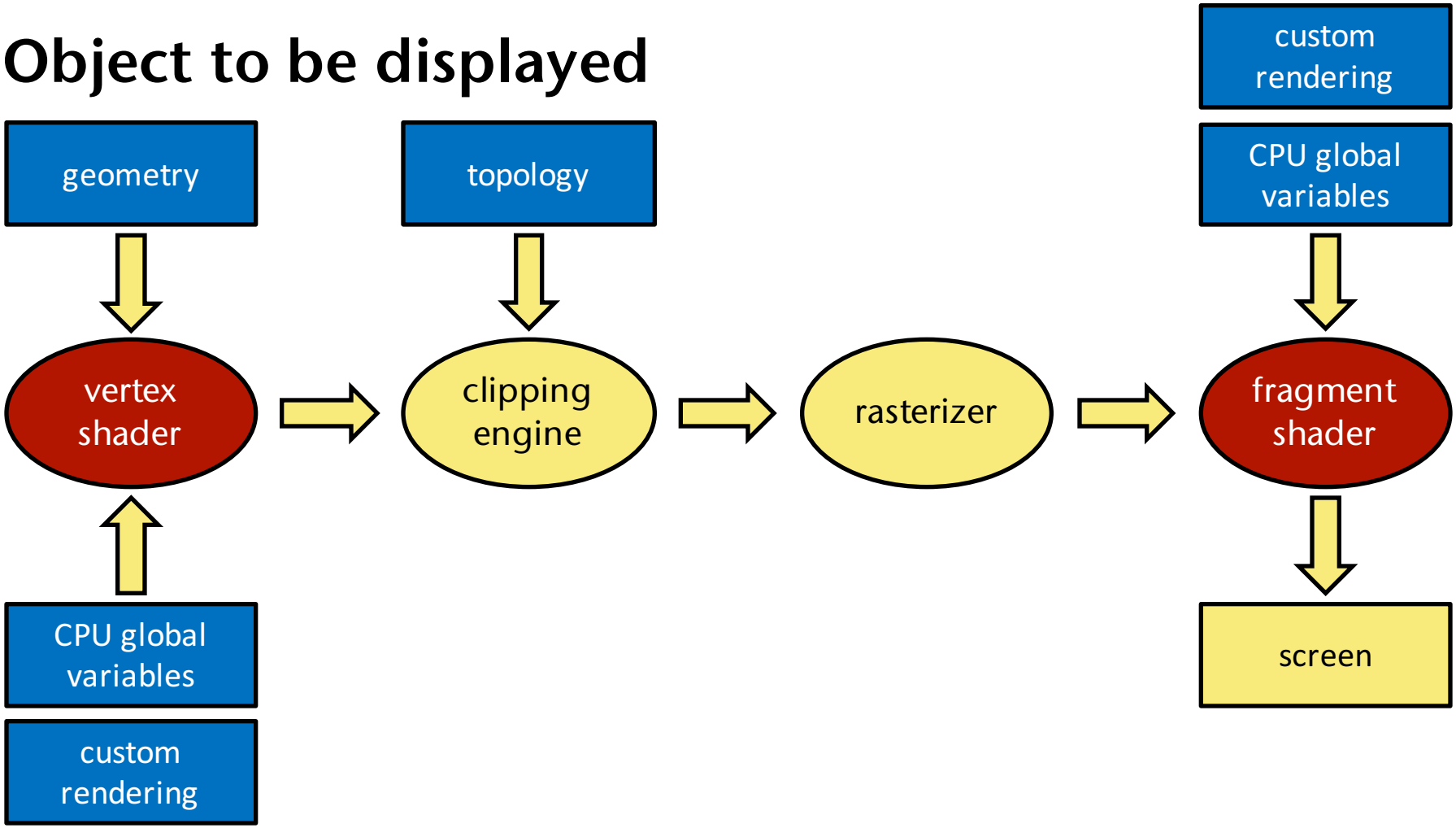


Screen **Normalized** **View**

Overview

- Introduction to 3D graphics
- Matrices and algebra
- *GPU programming*

OpenGL pipeline



Shader programming

- Vertex shader
- Fragment shader

- Each shader is
  - Written in C syntax, stored in a text file
  - Loaded with `loadVertexShader/loadFragmentShader`
  - The GPU driver compiles the code
  - Set up with `glAttachShader`

- GPU program is
  - Sent to the GPU with `glLinkProgram`
  - Activated with `glUseProgram`

Shader programming

- Vertex shader
  - handle vertex transformations and set `gl_Position`
  - access to standard OpenGL state variables with `gl_*`
  - declare variables as attributes (`varying` variables) so that they get linearly interpolated by the rasterizer
- Fragment shader
  - set fragment's (pixel) color and set `gl_FragColor`
  - access interpolated attributes (`varying` variables)

Shader setup with QT

```
QOpenGLShaderProgram program;

QOpenGLShader vert(QOpenGLShader::Vertex);
vert.compileSourceCode(<code_vert>);

QOpenGLShader frag(QOpenGLShader::Fragment);
frag.compileSourceCode(<code_frag>);

program.addShader(&vert);
program.addShader(&frag);
program.link();
```

Shader usage with QT

```
program.bind()
...
program.release()

Gluint loc = program.uniformLocation(<loc>);
program.setUniformValue(loc,<value>);
```

Basic shader/C syntax

- Types
  - `int, float, vec2, bool`
- Variable declaration `int x;`
- Variable assignment `x=5;`
- Arrays `int v[40]`
- Array access `v[3]=40;`
- Mathematics operations
  - `+, -, *, /`

- For loop `for(int i=0;<n;++i) {}`
- Conditional statements `if(x<n) {} else {}`

Basic shader

- Vertex shader

```
void main(){
   gl_FrontColor  = gl_Color;
   gl_Position  =
     gl_ModelViewProjectionMatrix  * gl_Vertex;
}
```

- Fragment shader

```
void main(){
     gl_FragColor  = gl_Color;
}
```