3D Computer Graphics and Animation

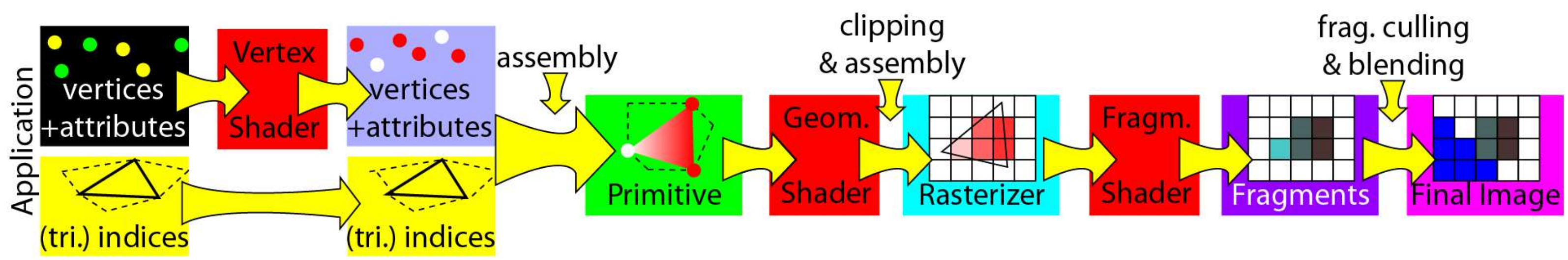
Graphics Pipeline
Time to take some shade

Elmar Eisemann

Delft University of Technology

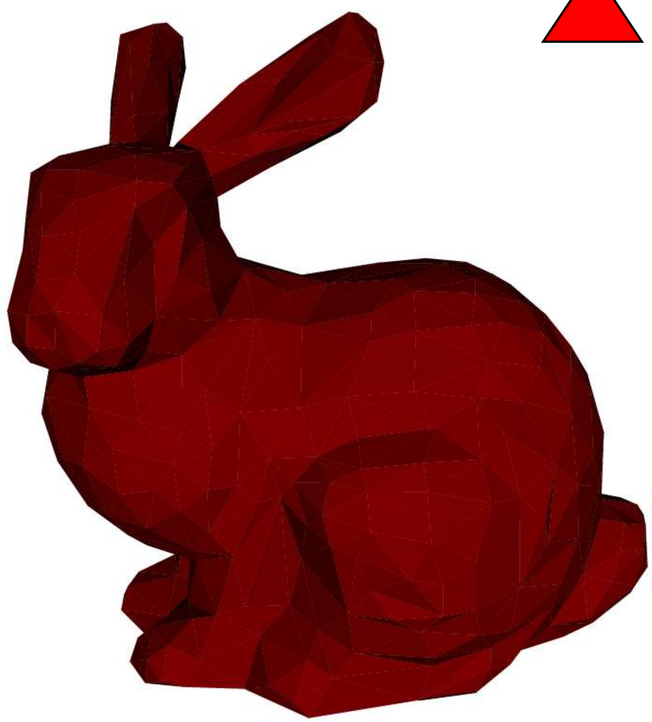
Rasterization

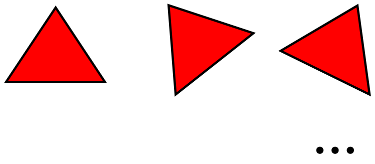
Rasterization via the Graphics Pipeline



Simplified Graphics Pipeline

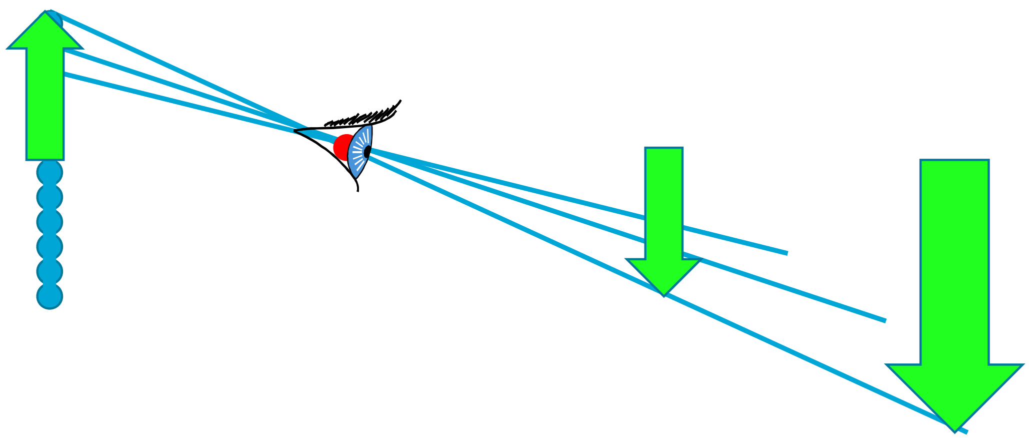
- Models are typically lists of triangles





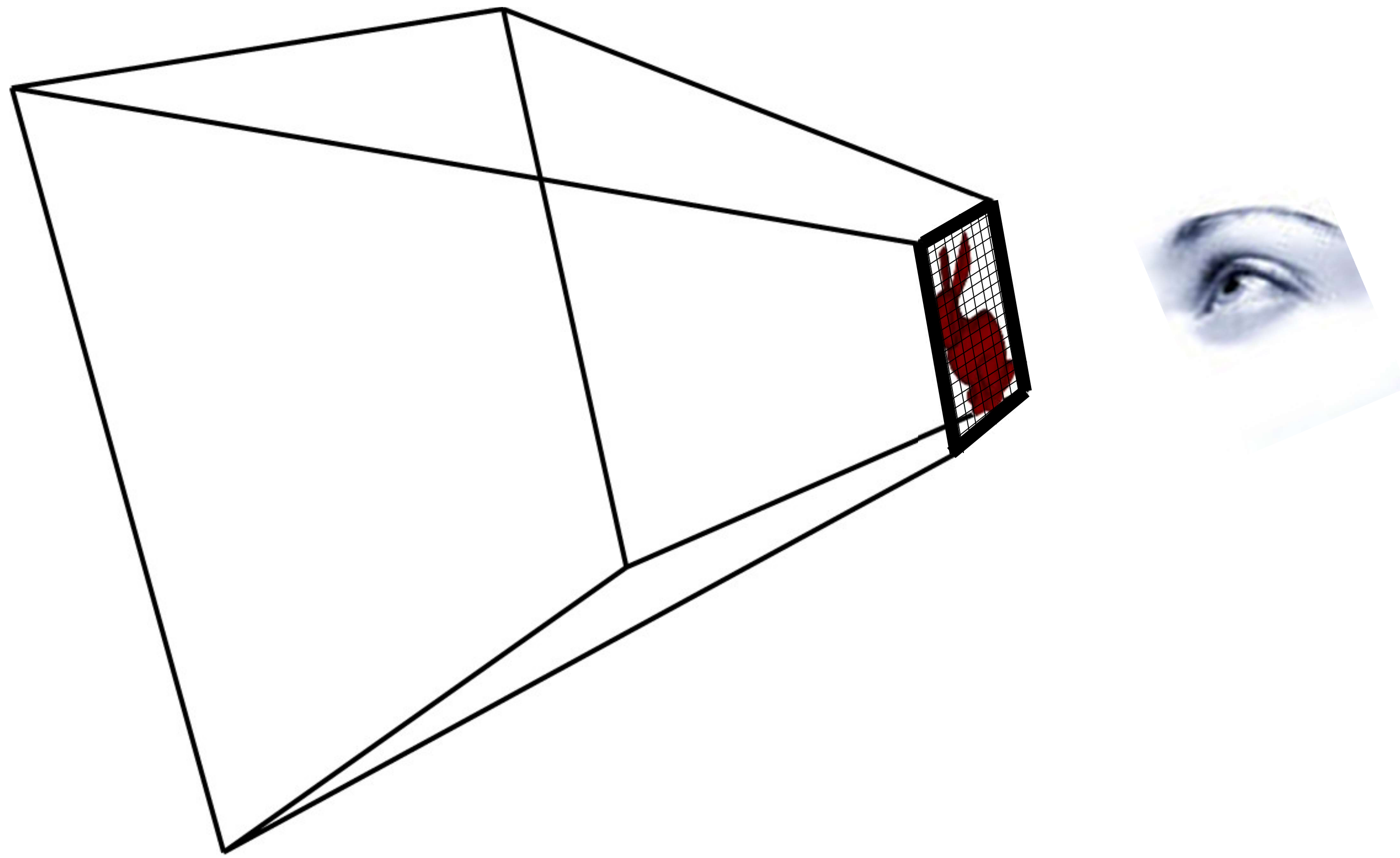
Virtual Camera

- Camera Plane in front of the eye



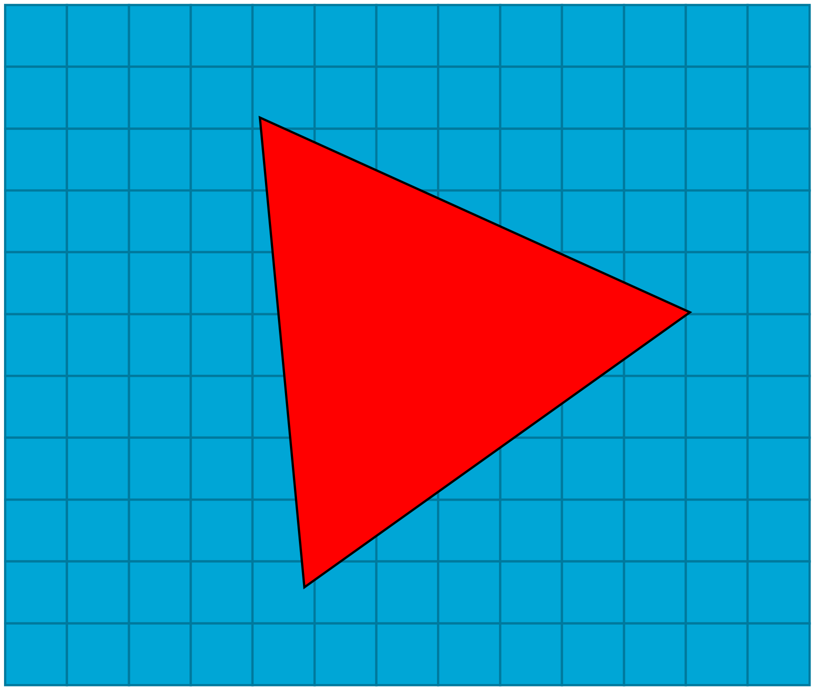
Simplified Graphics Pipeline

- Projection: Transform coordinates to screen



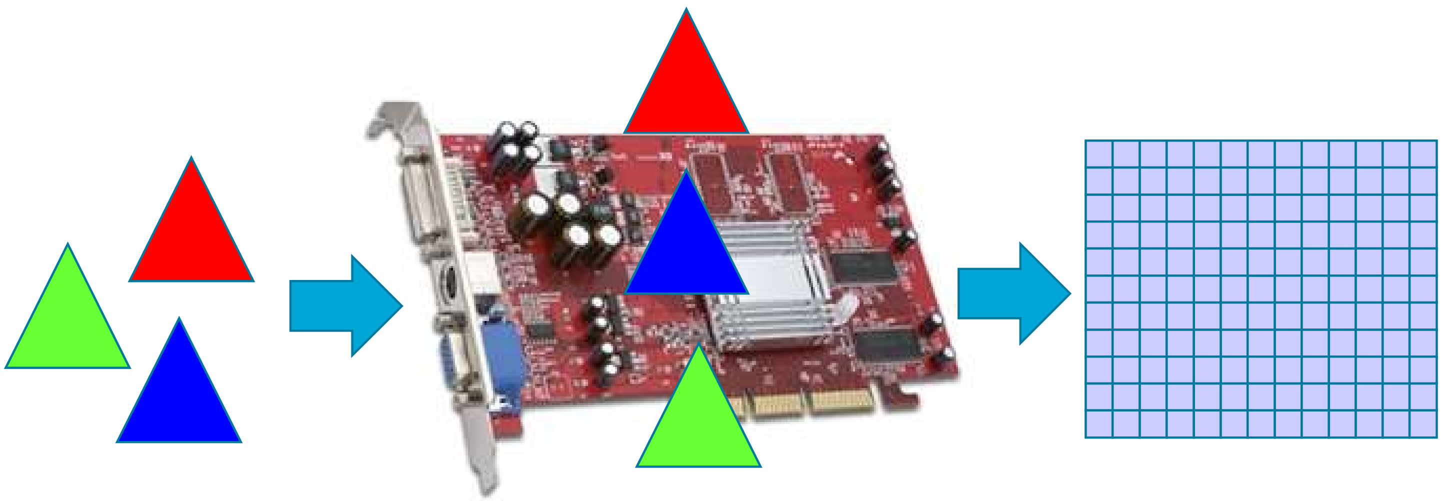
Simplified Graphics Pipeline

- Rasterization: Fill screen pixels



Simplified Graphics Pipeline

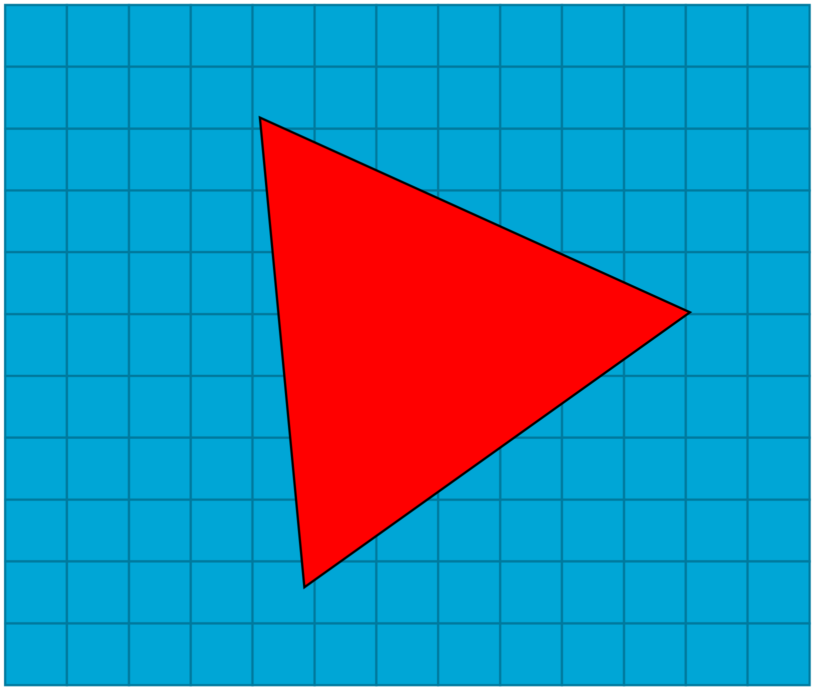
- Highly parallelizable GPUs



…NVIDIA RTX 3080 Ti has around 10240 cores…

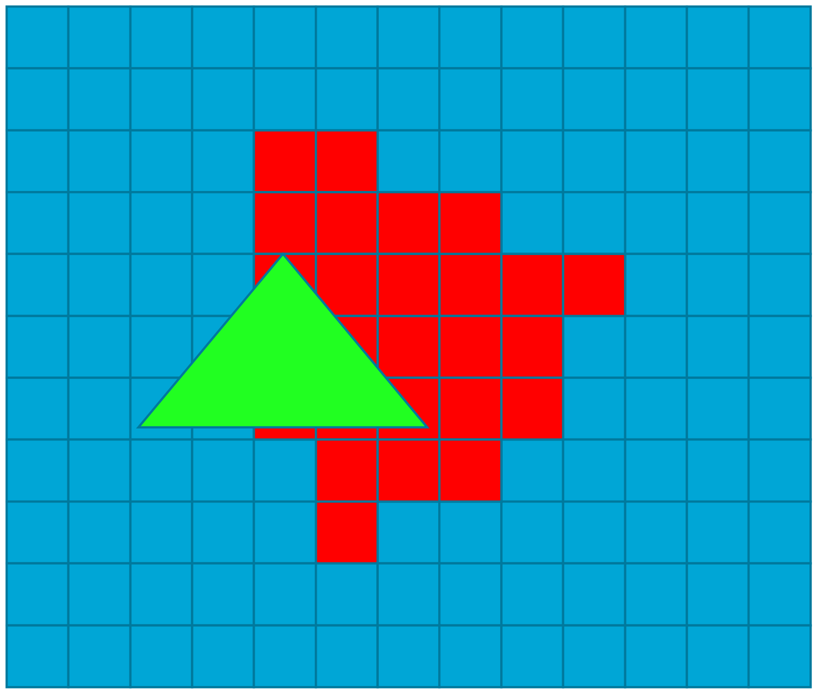
Simplified Graphics Pipeline

- Catch: Let’s look at a second triangle…



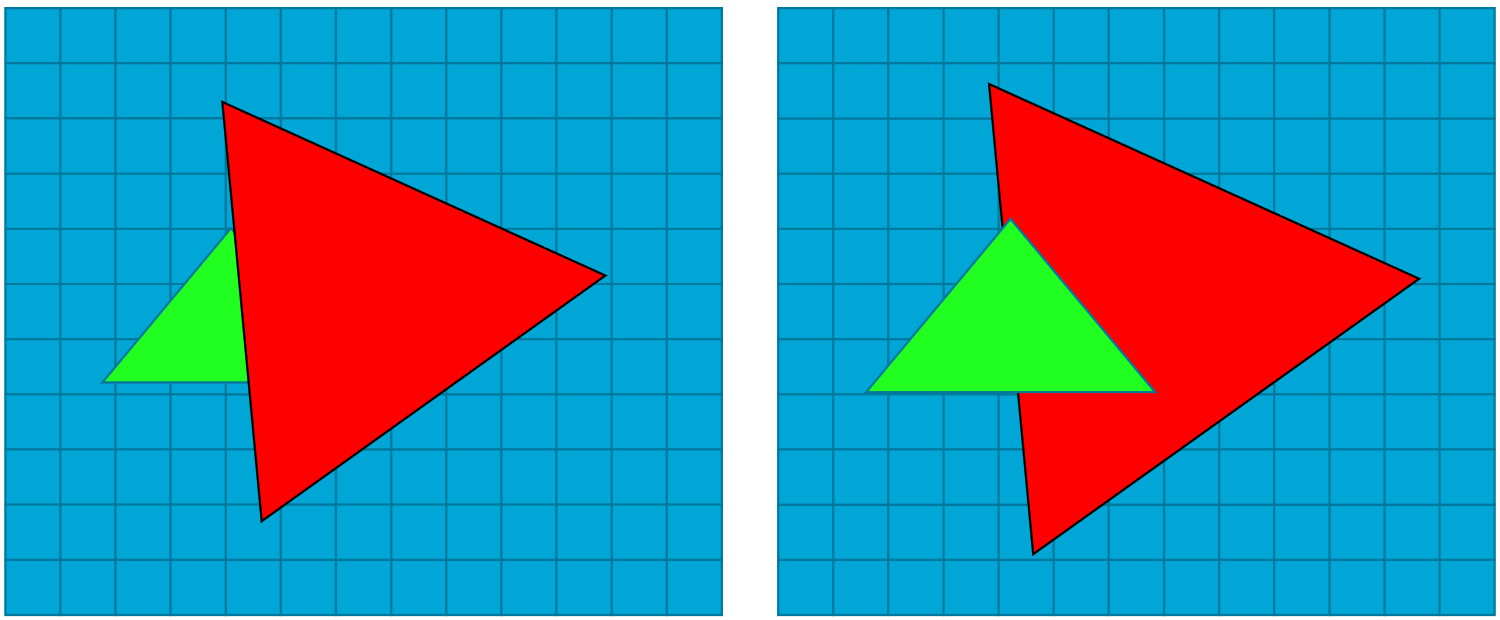
Simplified Graphics Pipeline

- Catch: Let’s look at a second triangle…



Simplified Graphics Pipeline

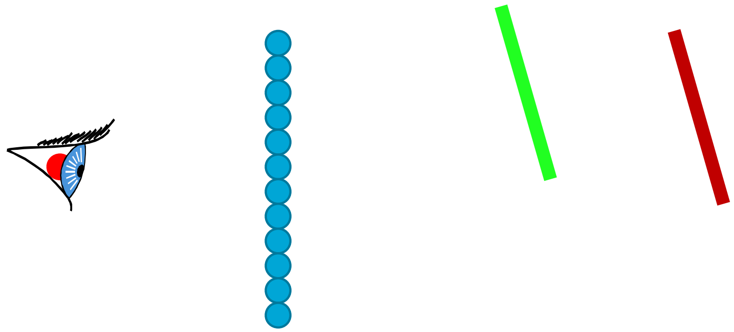
- Catch: Triangle drawing order changes result



As for ray tracing: we need to know the closest triangle in a pixel

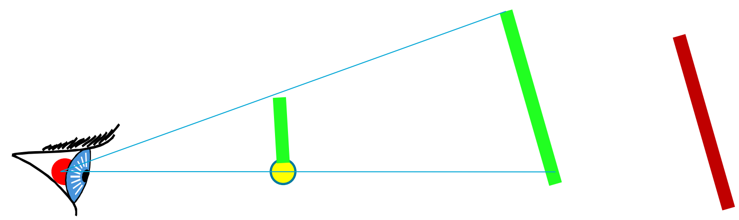
Simplified Graphics Pipeline

- Depth Test: Avoid sorting!
- Store a depth in each pixel



Simplified Graphics Pipeline

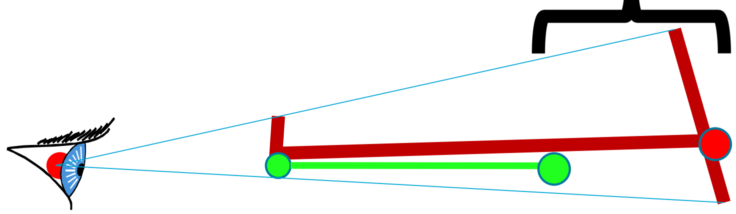
- Depth Test: Avoid sorting!
- Store a depth in each pixel



Simplified Graphics Pipeline

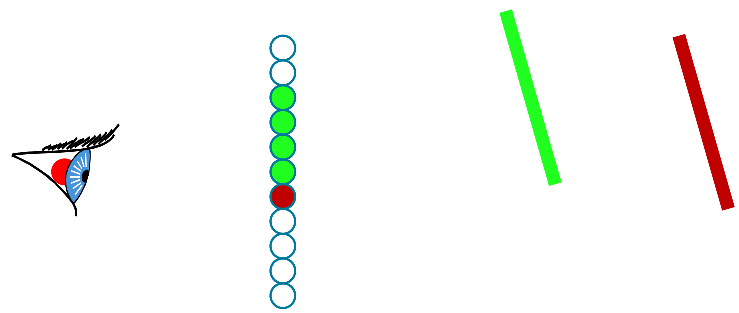
- Depth Test: Avoid sorting!
- Store a depth in each pixel

Compare new distance to stored distance
Update pixel only if new distance is nearer



Simplified Graphics Pipeline

- Depth Test: Avoid sorting!
- Store a depth in each pixel

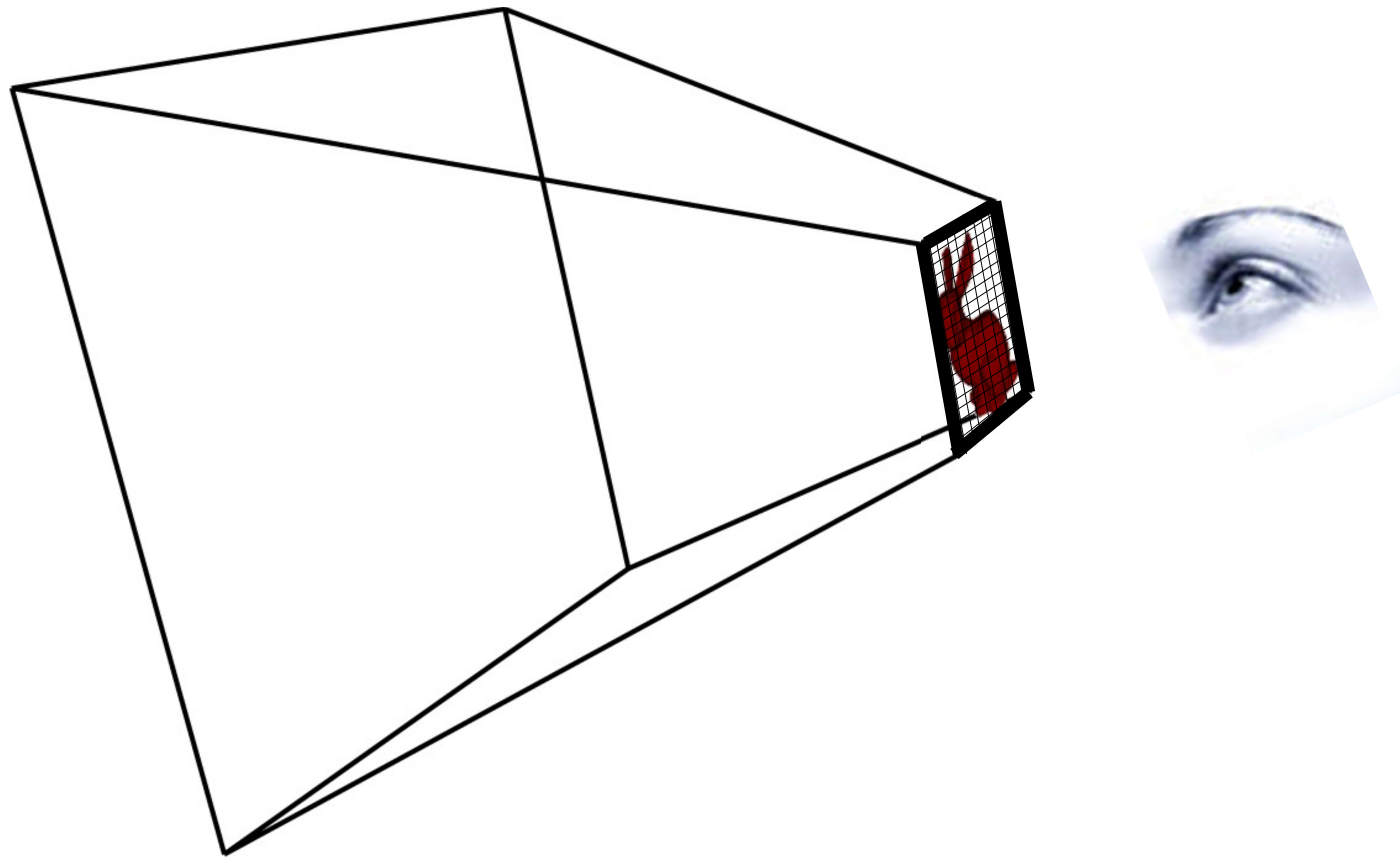


Creating a virtual camera model

- Given a 3D point, we should find a function that results in the point's projection in the photo.

Simplified Graphics Pipeline

- Projection: Transform coordinates to screen



Virtual Camera Model

Projecting a scene point with the camera:

- Apply camera position (adding an offset)
- Apply rotation (matrix multiplication)
- Apply projection (non-linear scaling)

Our camera starts to become complicated

and not well adapted to a hardware solution…

There has to be a better way…

Homogenous Coordinates - Definition

- N-D *projective space* P^n
  is represented by N+1 coordinates, has no null vector,
  but a special equivalence relation:

Two points p, q are **equal**

iff (if and only if)

exists a!=0 such that p*a = q

Examples in a 2D projective space P^2:
(2,2,2) = (3,3,3) = (4,4,4)=(π, π, π)
(2,2,2) $\neq$ (3,1,3)
(0,1,0) = (0,2,0)
(0,0,0) not part of the space

Homogeneous Coordinates

To embed a standard vector space R^n in an n-D projective space P^n, we can map:

$(x_0,x_1,\ldots x_{n-1})$ in R^n to $(x_0,x_1,\ldots x_{n-1}, 1)$ in P^n

Typically, the last coordinate in a

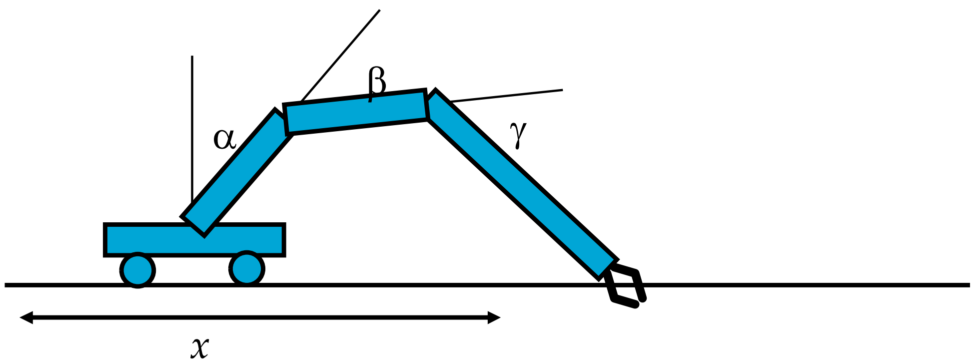
projective space is denoted with w.

Homogeneous Coordinates

- We have seen that many operations in R^n can be represented by matrices in P^n
- Transformations can be concatenated by multiplying the matrices together

Complex Objects

- Objects are often defined via many components
  - E.g., wheels of cars, fingers on hands on arms …
- Concatenate matrices to place objects relative to each other
  - Changing one matrix affects everything hereafter



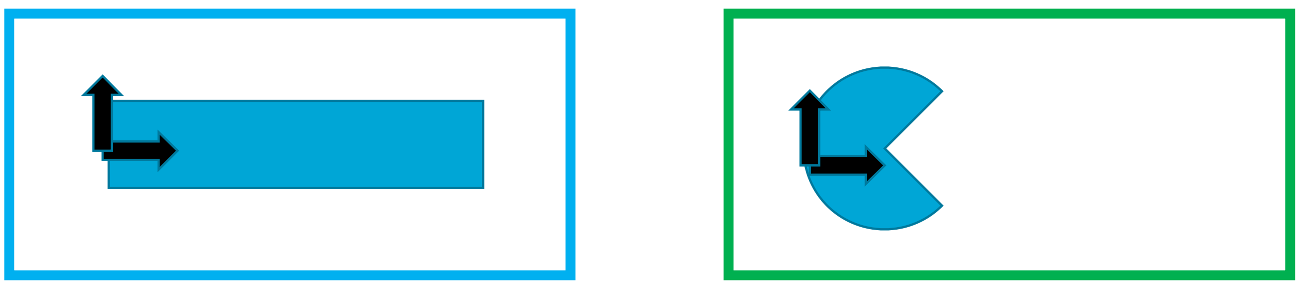
Complex Objects

Example:

We make a “complex” robot arm consisting of two parts:

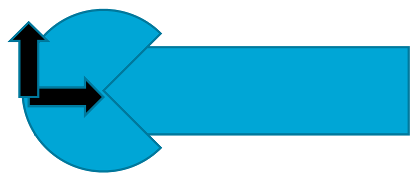
The arm itself and a hand

Both are designed independently
and are at the origin (shown below)



Complex Objects

- Using the coordinates as is, you get:



- That does not look right!
- Instead: Let’s define a matrix to shift the objects to the wanted relative location

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)

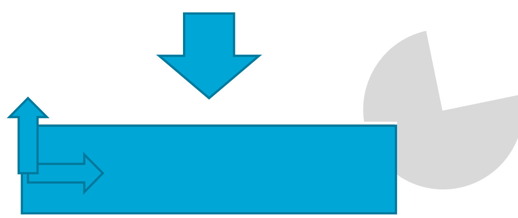
Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm

Resulting positions
After applying the object
vertices to matrix S



Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T

Complex Objects

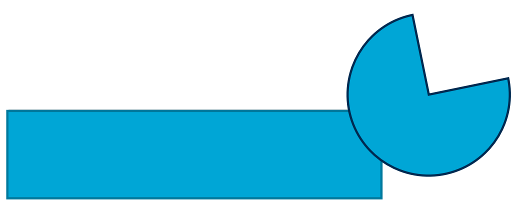
- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand

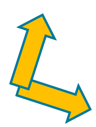


Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - **S:=SN, with N a new rotation matrix**
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand

Complex Objects

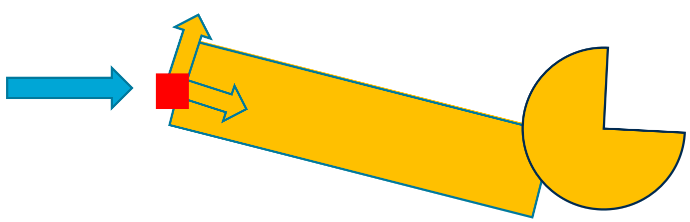
- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - **S:=SN, with N a new rotation matrix**
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand



Complex Objects

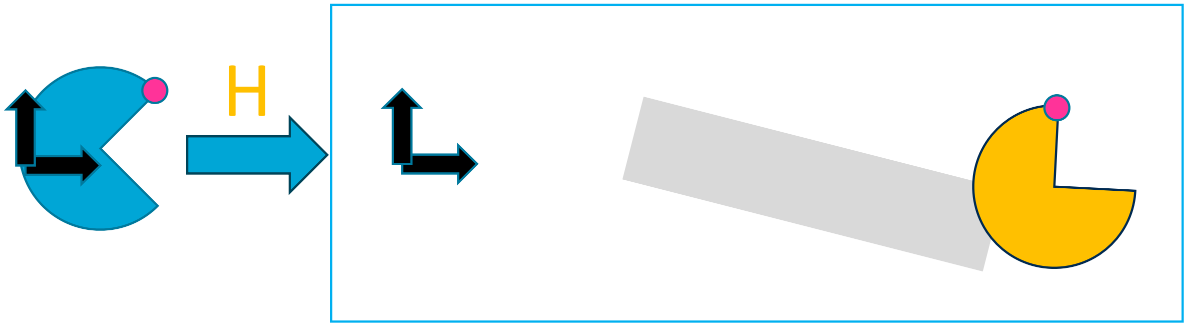
- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - **S:=SN, with N a new rotation matrix**
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand

Same position



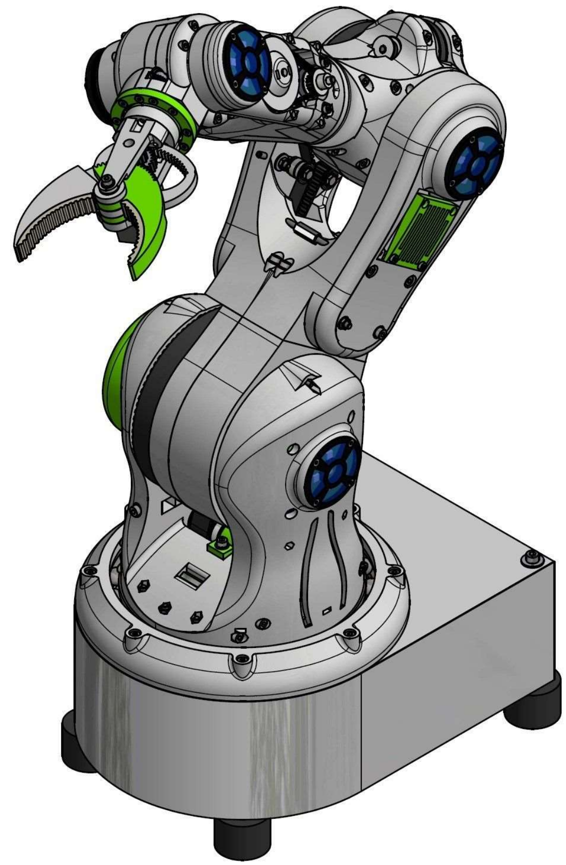
Complex Objects

We found a matrix H to transform the vertices from the original hand definition into the wanted position relative to the arm!



Let’s try it out!

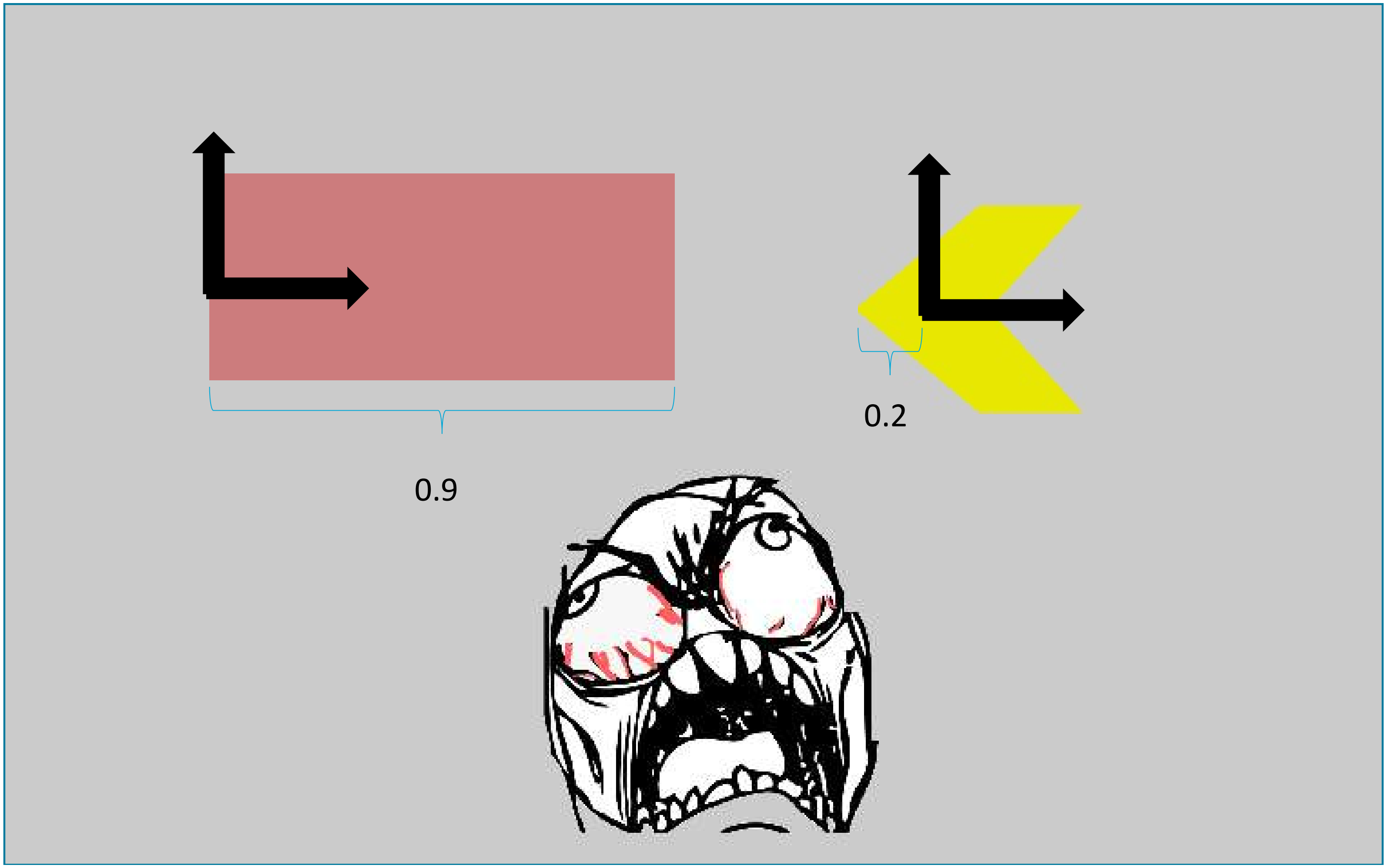
- Professional Robot Arm:



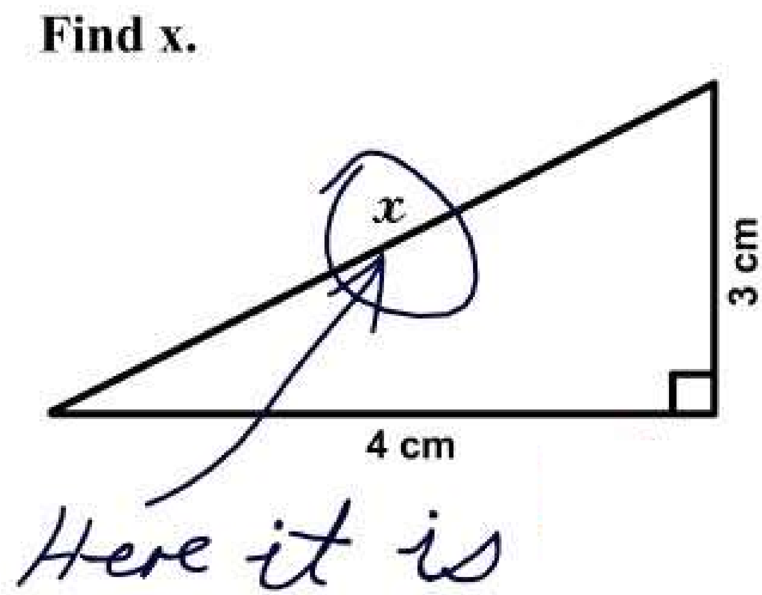
Let's try it out!

- Professional Robot Arm:

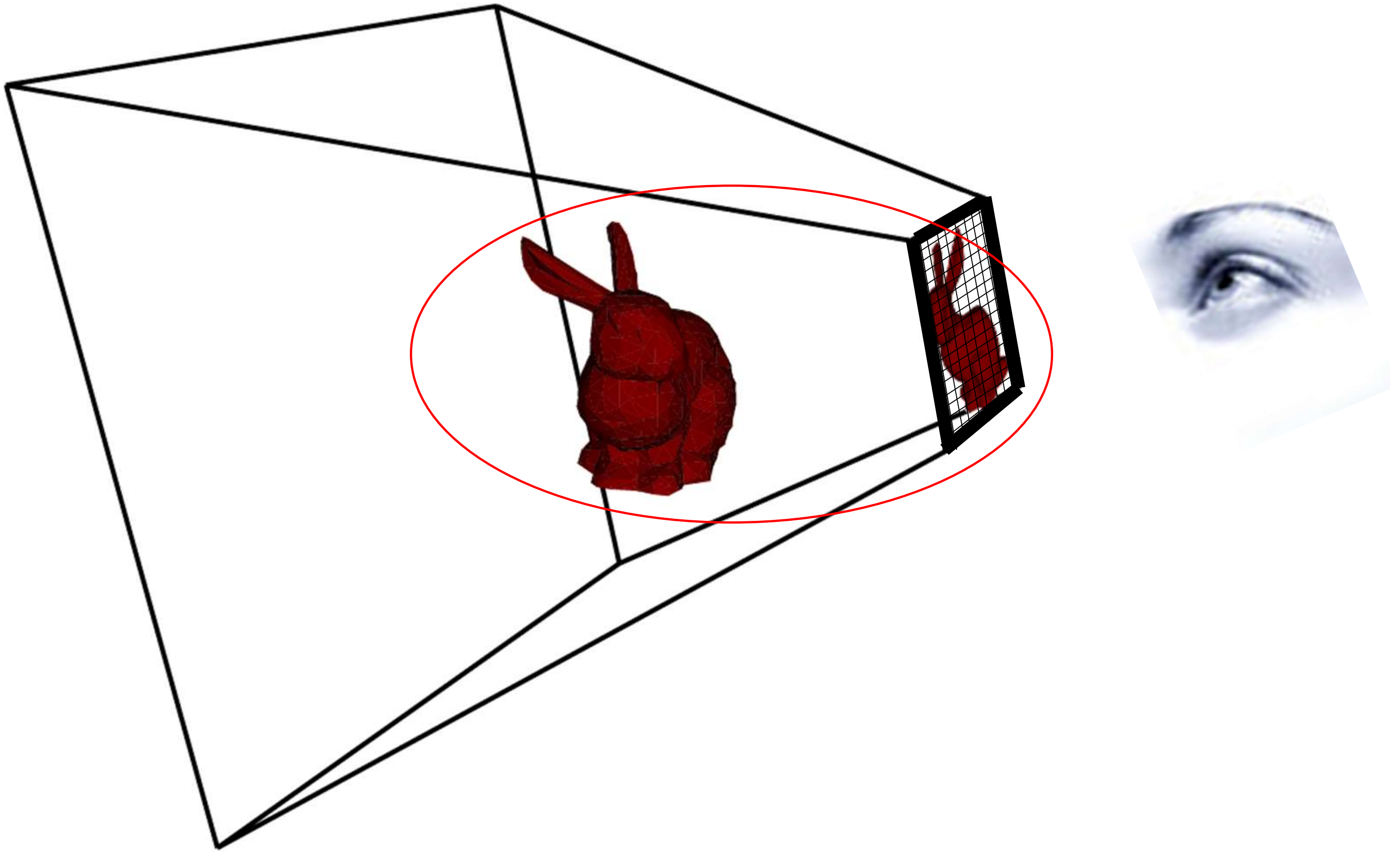
Let's try it out!



Questions?

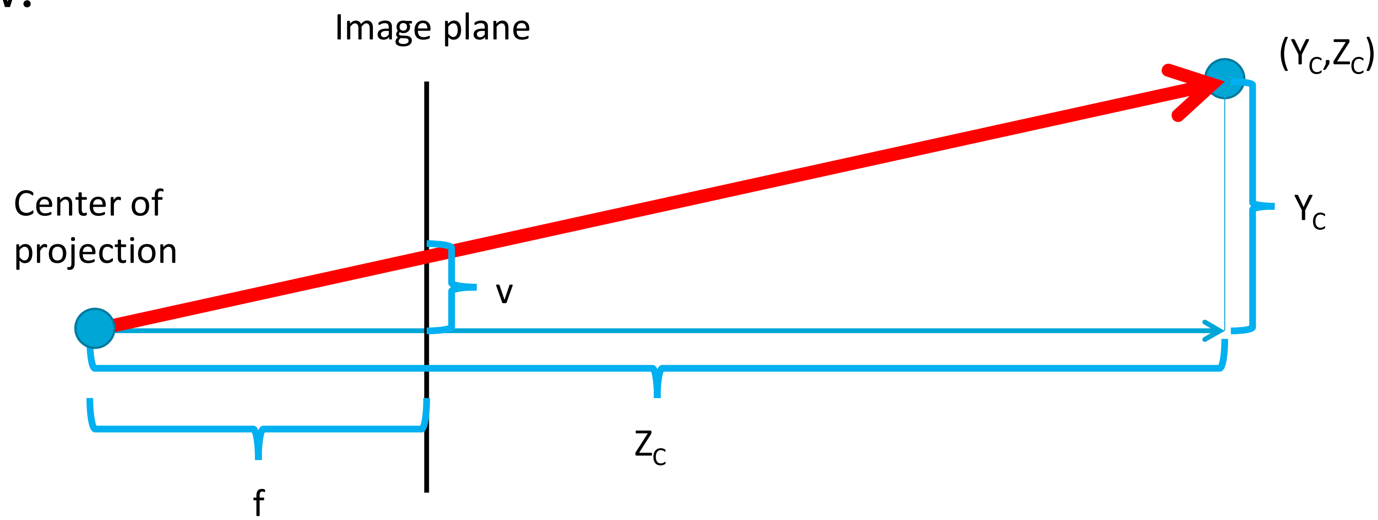


One pipeline step is still missing...



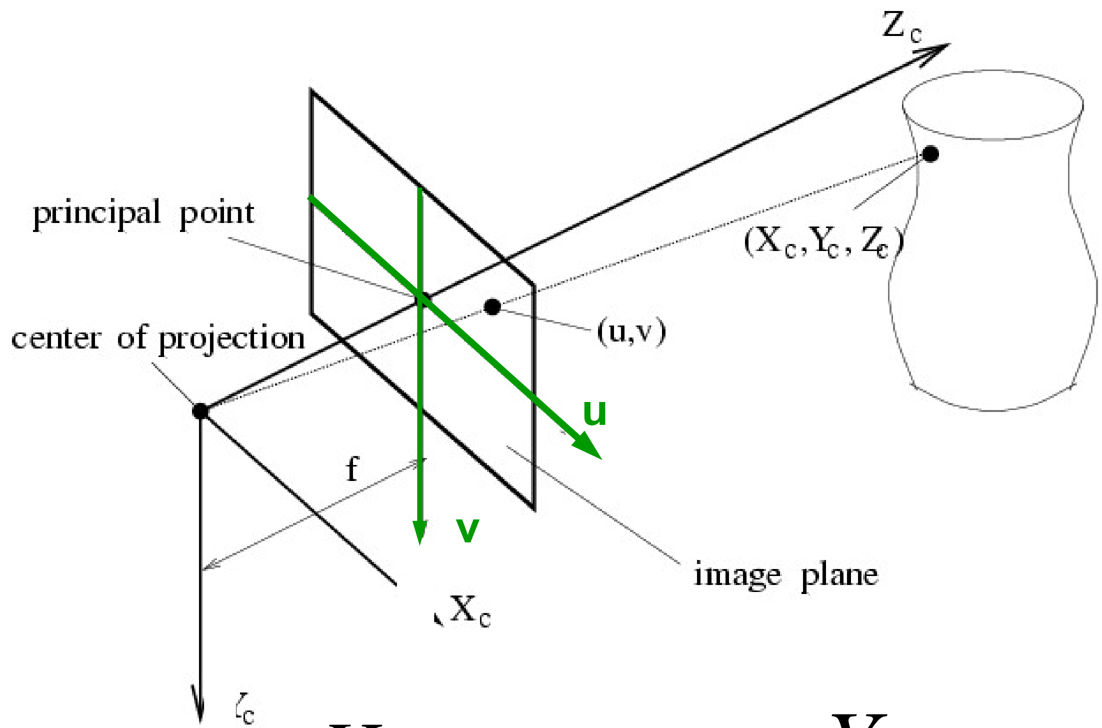
Perspective Projection

- sideview:



Similar triangles: $v / f = Y_C / Z_C$

Perspective Projection



$$u = f\frac{X}{Z} \qquad v = f\frac{Y}{Z}$$

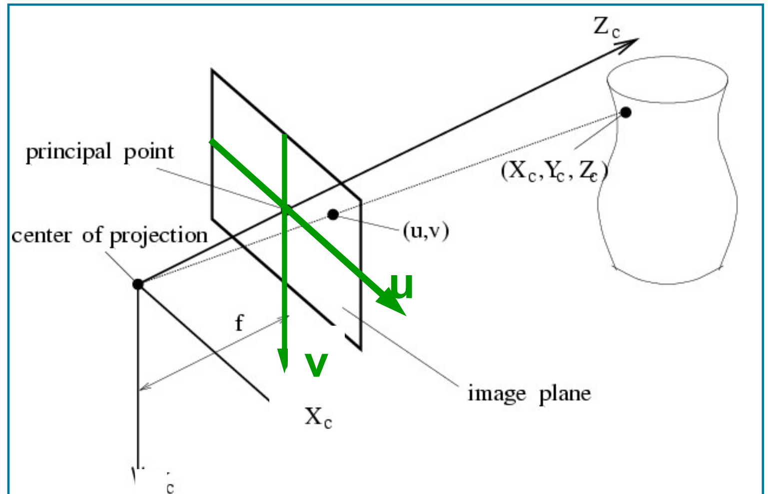
Perspective Projection

- “Homogenize” the points

$$P = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad \Rightarrow \quad \widetilde{P} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- Look for M such that:

$$M\widetilde{P} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix}$$



$$u = f\,\frac{X}{Z} \qquad v = f\,\frac{Y}{Z}$$

Perspective Projection

- Hint: Think projective!

$$M\tilde{P} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix} \Rightarrow M\tilde{P} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix}$$

Perspective Projection

- Solution:

$$M = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

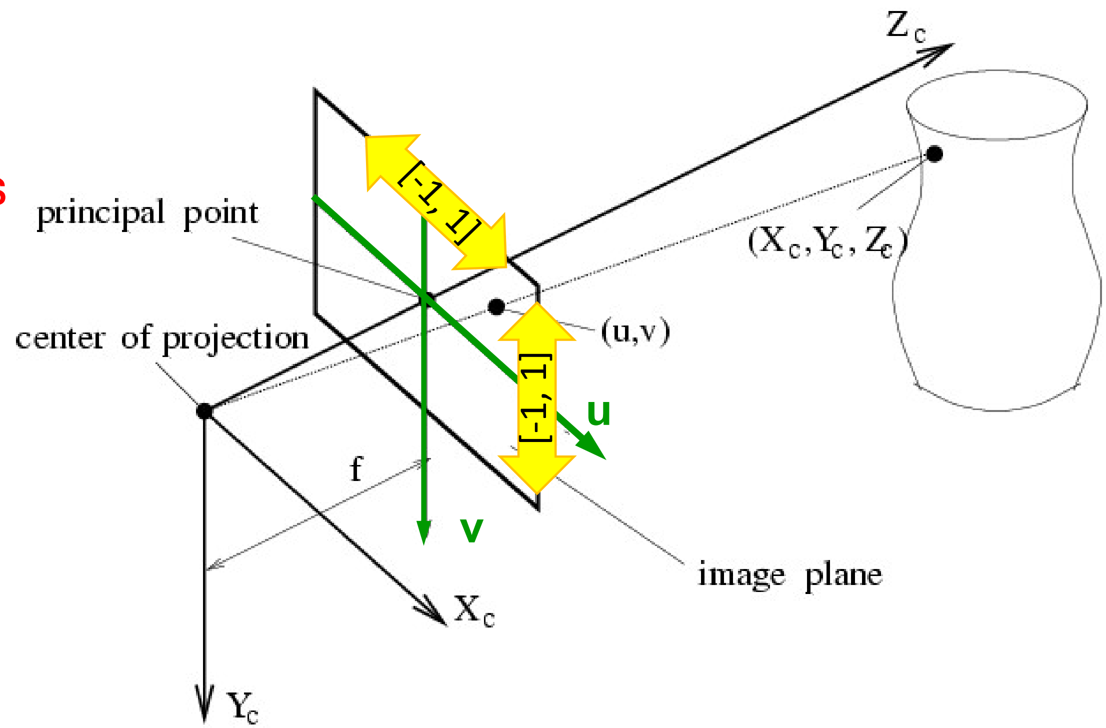
Perspective Projection

- Solution:

$$M = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Adding Image Space to Camera Model

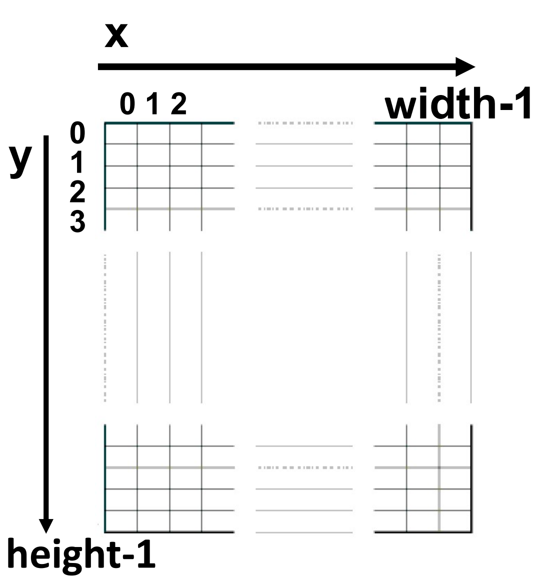
- Transform u, v
  into pixel positions



$$u = f\frac{X}{Z} \qquad v = f\frac{Y}{Z}$$

Internal Camera Parameters

- $(-1,1)^2 \Rightarrow (0, \text{width}-1) \times (0, \text{height}-1)$
- Homework: Find a matrix to do this mapping

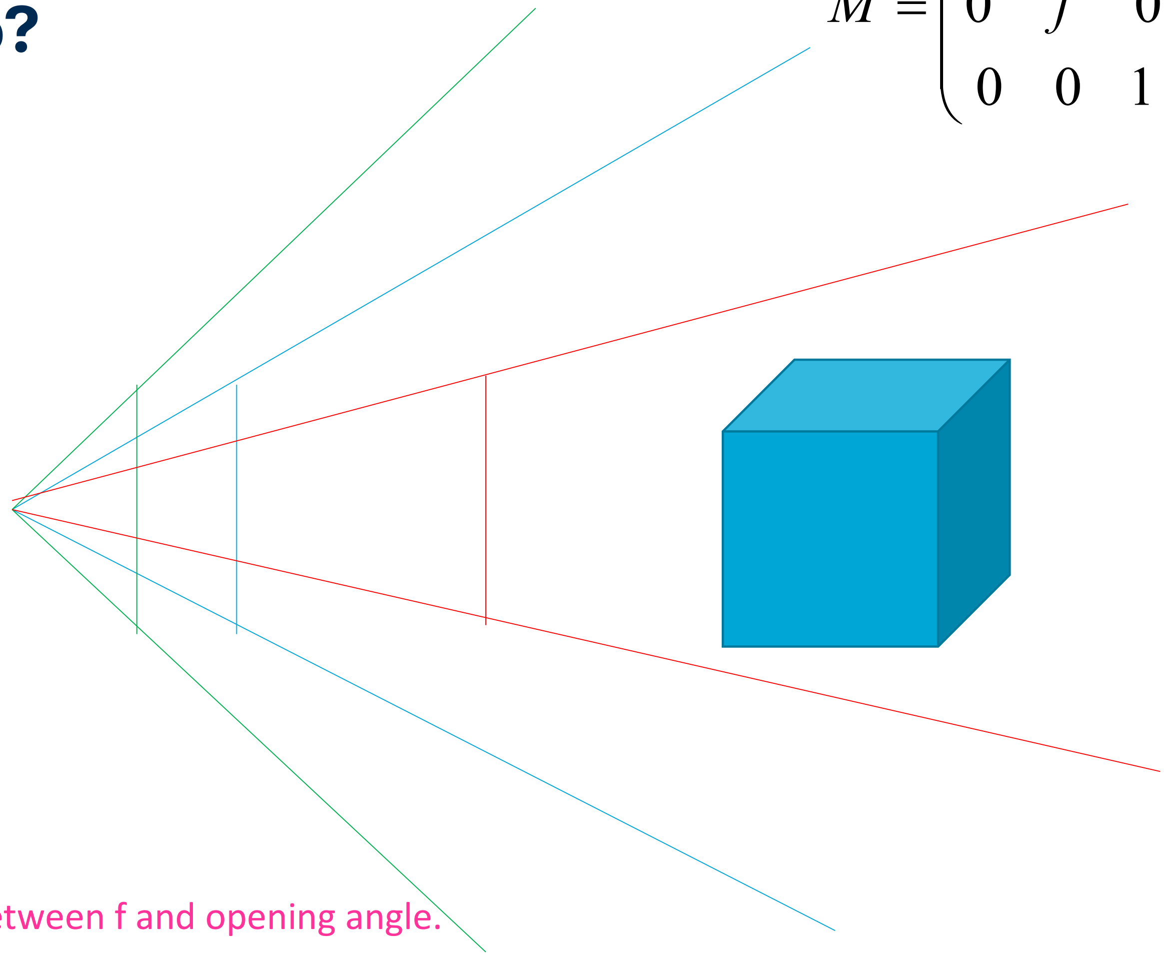


$$\begin{cases} x = k_x u + x_0 \\ y = k_y v + y_0 \end{cases}$$

$$\begin{pmatrix} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

What does f do?

$$M = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$



Nice exercise:
Find the link between f and opening angle.

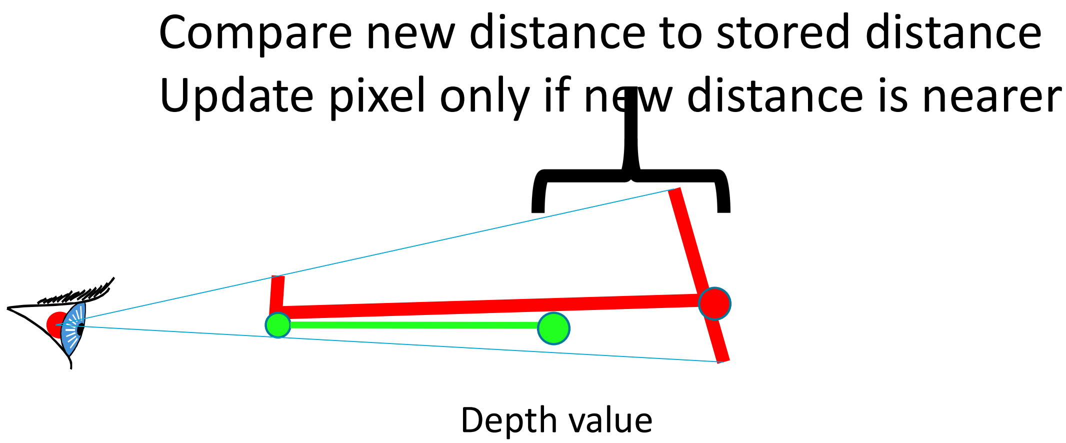
It could have been so simple...

- What is the problem of this matrix for the Graphics Pipeline?

$$M = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

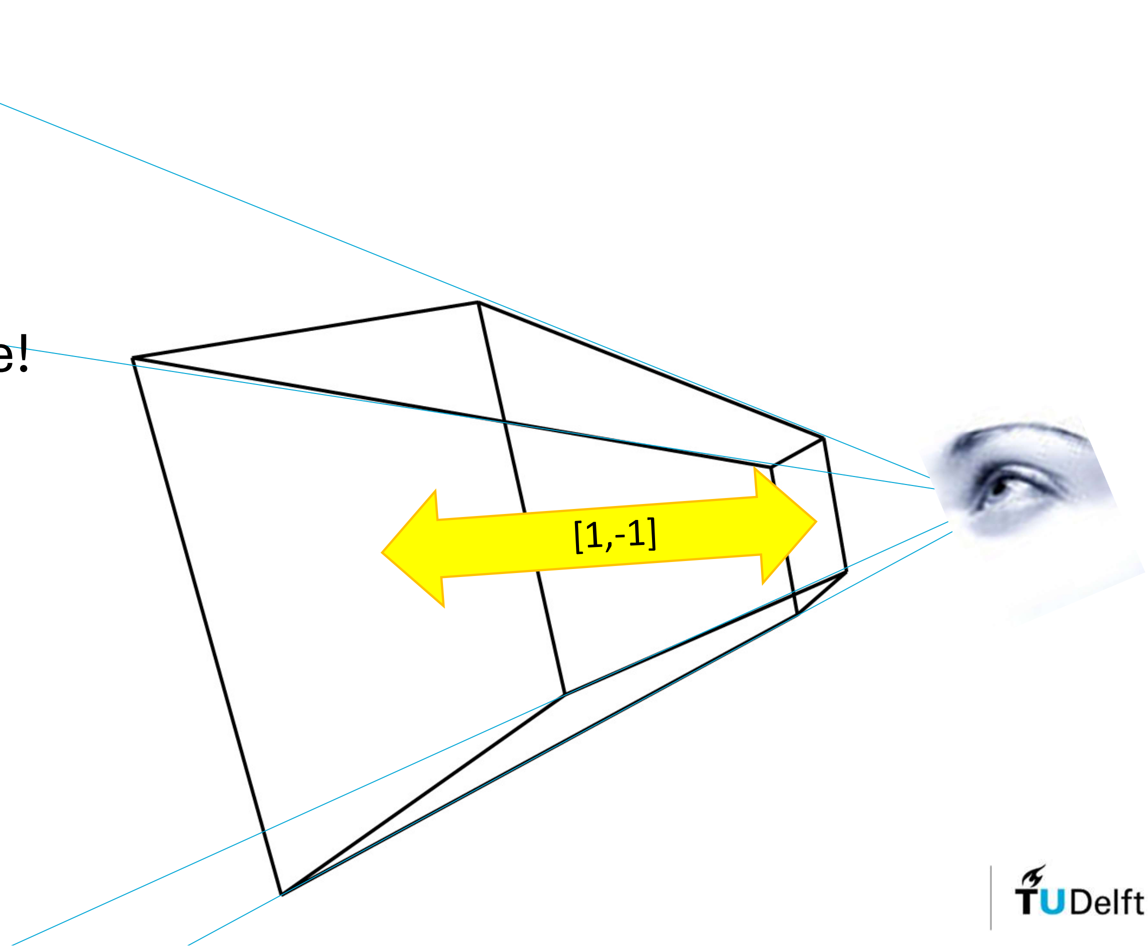
We need a depth value!

- We need to keep a Z coordinate!



Problem:

- A 3D scene is infinite…
- How do we represent Z?
- Solution:
  Add a near and far clipping plane!



TUDelft

Camera in Matrix Representation

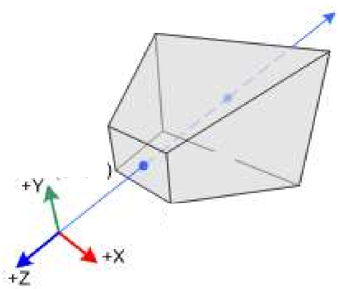
- Projection matrix:

$$
\begin{array}{l}
\text{screen mapping} \\
\text{screen mapping} \\
\\
\\
\end{array}
\begin{bmatrix}
f & 0 & 0 & 0 \\
0 & f & 0 & 0 \\
\multicolumn{4}{c}{\text{New line – mapping the z value}} \\
0 & 0 & 1 & 0
\end{bmatrix}
$$

Camera in Matrix Representation

- Projection matrix:

$$\begin{array}{l} \text{screen mapping} \\ \text{screen mapping} \\ \\ \\ \end{array} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ \multicolumn{4}{c}{\text{New line – mapping the z value}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

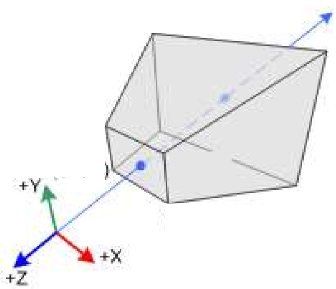


By convention, camera looks along the negative z-Axis
…don't bother too much about it…

Camera in Matrix Representation

- Projection matrix:

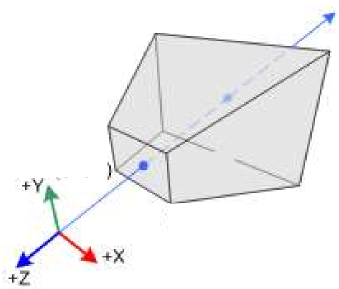
$$\begin{array}{r}\text{screen mapping}\\ \text{screen mapping}\\ \\ \\ \end{array}\begin{bmatrix} f & 0 & 0 & 0\\ 0 & f & 0 & 0\\ \multicolumn{4}{c}{\text{New line – mapping the z value}}\\ 0 & 0 & -1 & 0 \end{bmatrix}$$



Camera in Matrix Representation

- Projection matrix:

$$\begin{array}{r} \text{screen mapping} \\ \text{screen mapping} \\ \\ \\ \end{array} \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ \multicolumn{4}{c}{\text{New line – mapping the z value}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



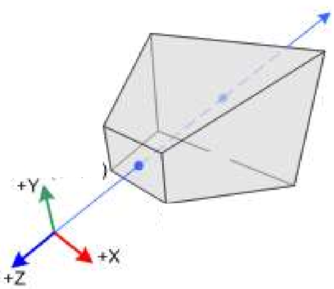
16:9

4:3

Camera in Matrix Representation

- Projection matrix:

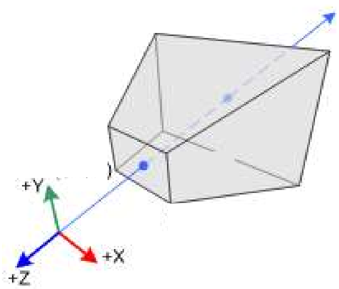
$$
\begin{array}{r c l}
\text{screen mapping} & \left[\begin{array}{cccc} \dfrac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \dfrac{near+far}{near-far} & \dfrac{2near\,far}{near-far} \\ 0 & 0 & -1 & 0 \end{array}\right] & \\
\text{screen mapping} & & \\
 & & \text{Z mapping}
\end{array}
$$



Camera in Matrix Representation

- Projection matrix:

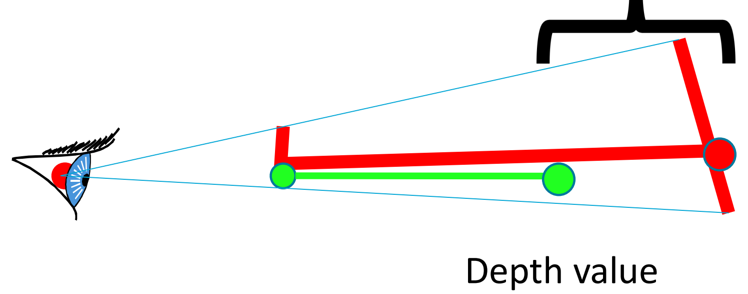
$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2near\,far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



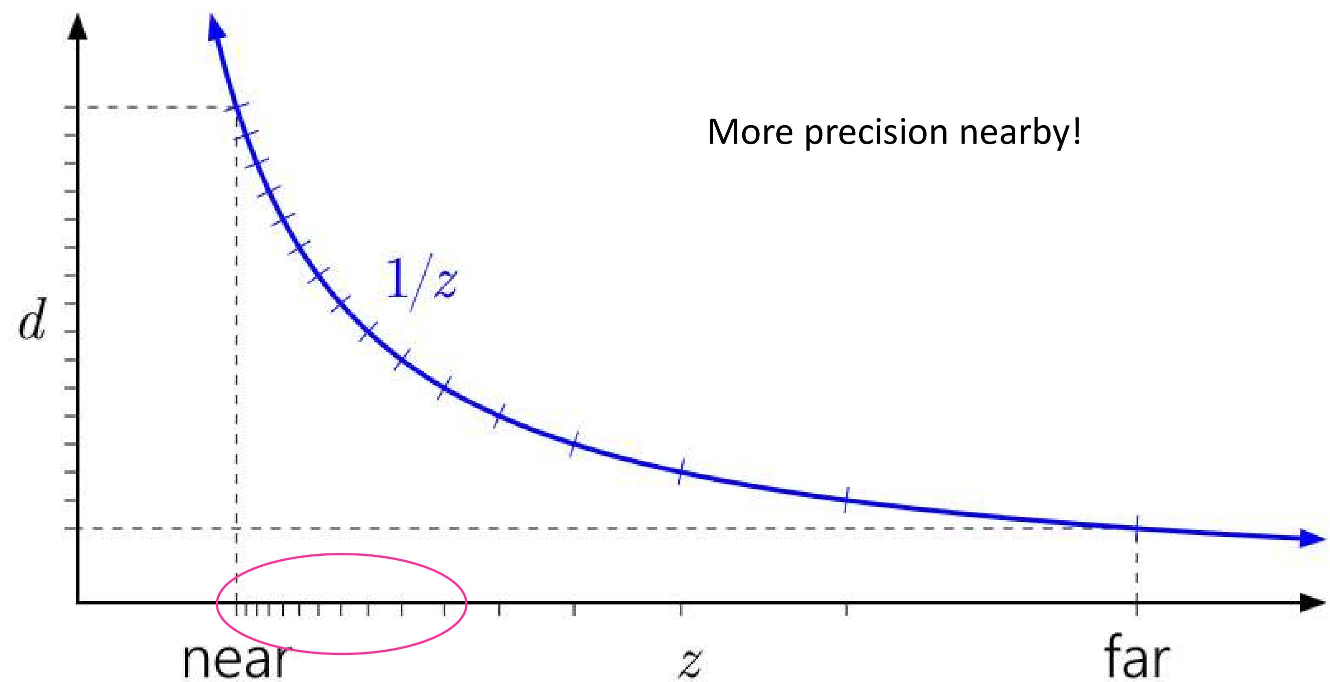
- Verify: (0,0,-near,1) maps to depth -1, and (0,0,-far,1) to 1
- What depth does (0,0,-(far+near)/2,1) map to? Z-Buffer is not linear!

Is z-Buffer non-linearity a problem?

Compare new distance to stored distance
Update pixel only if new distance is nearer



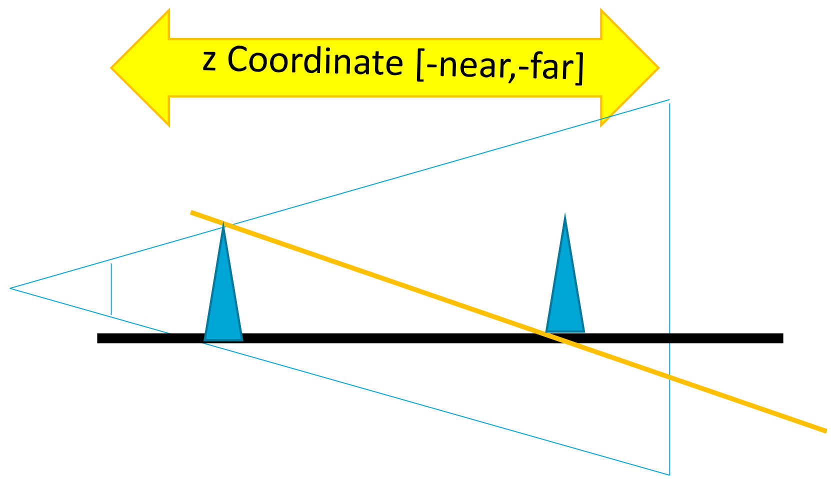
Z-Buffer Visualization



More precision nearby!

https://developer.nvidia.com/content/depth-precision-visualized

Illustration of the Camera Mapping



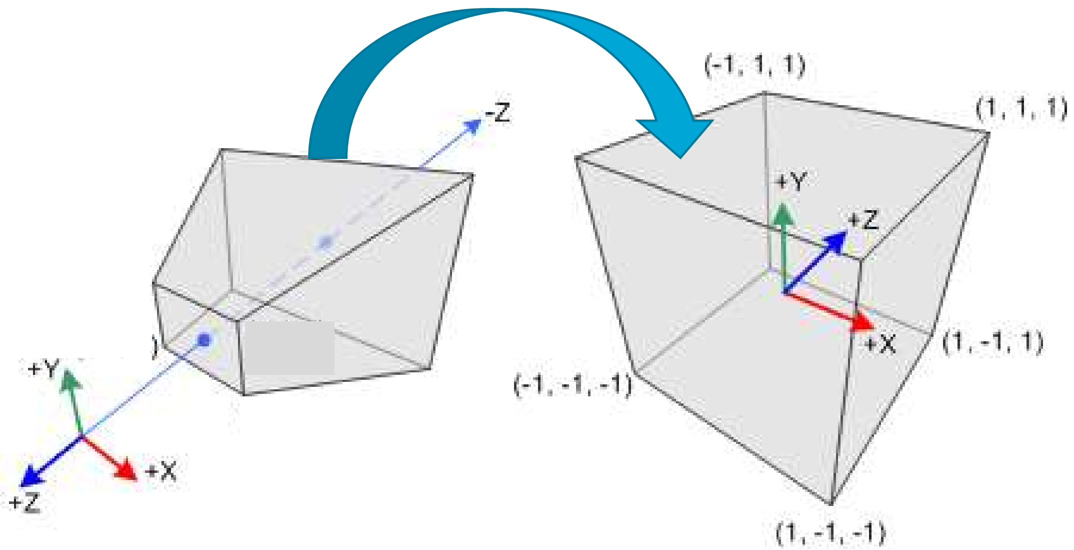
$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2near\,far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

After normalization
(division by w)

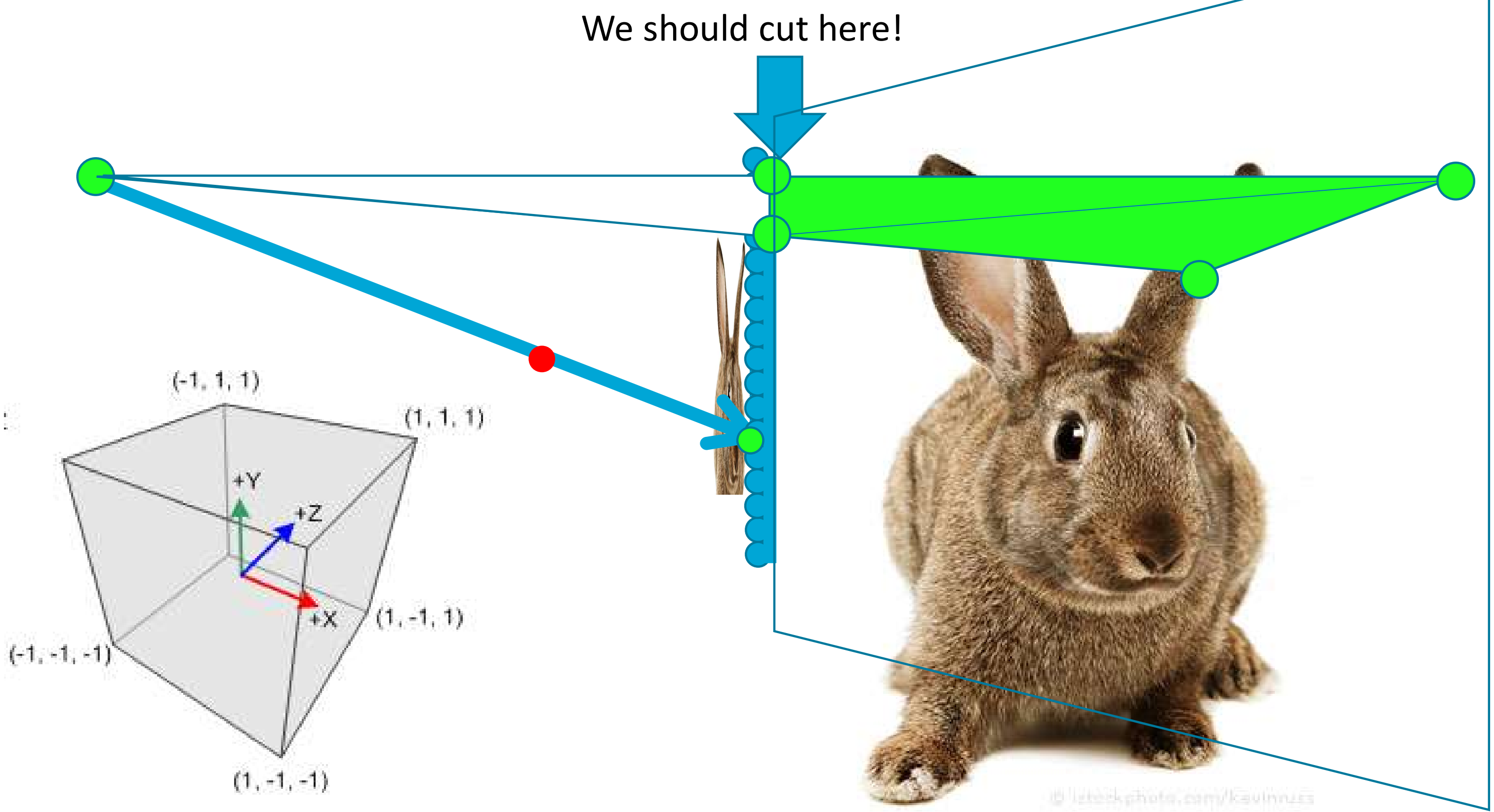
Camera Space

- Perspective Frustum is mapped on a cube

$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2near\,far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



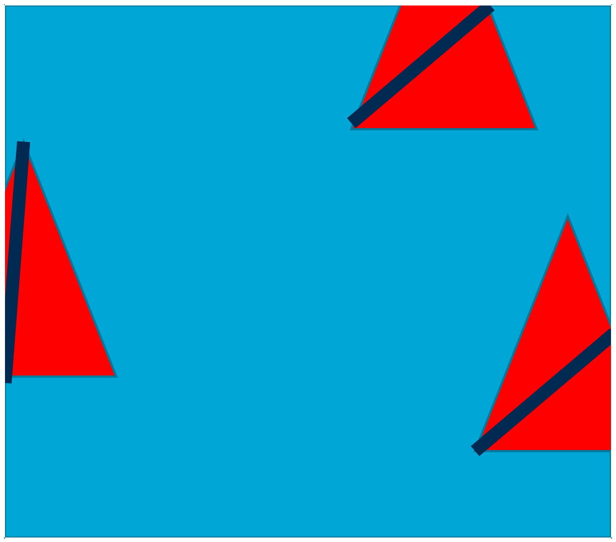
Final Step



- Test if triangle lies in the cube!

Clip Space

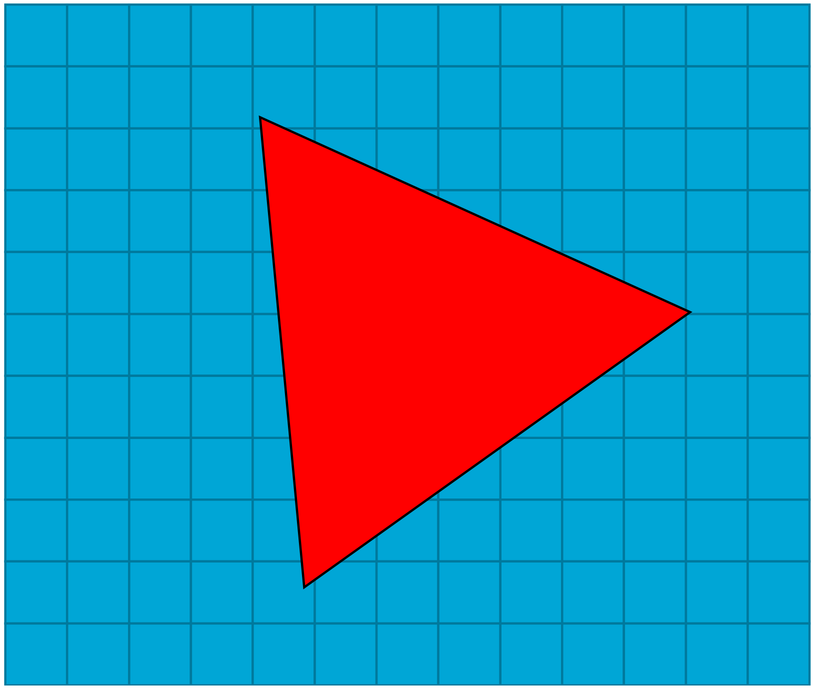
- Create clipped triangles



Last pipeline step: Rasterization

- Now, all primitives are clipped and projected on the screen.

- Pixel filling +
- Depth Buffer



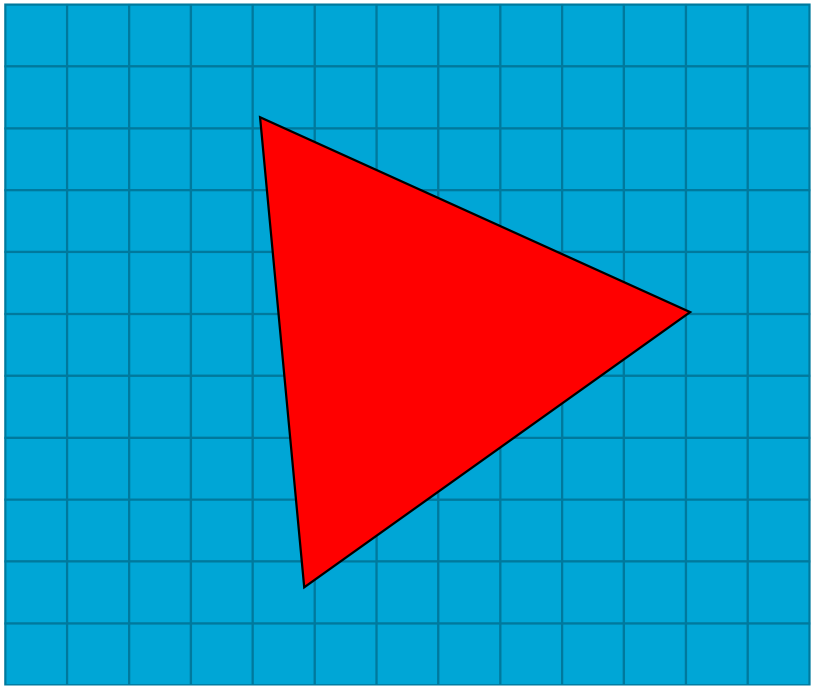
Last pipeline step: Rasterization

- Now, all primitives are clipped and projected on the screen.

- Pixel filling +
- Depth Buffer

Remark 1: Complex rules to avoid ambiguities, but if pixel center is inside the triangle, the pixel is filled

Remark2: The projection delivers depth d in [-1,1], d is stored using (d+1)/2, thus in [0,1]



Graphics Pipeline

Object Transformation

Viewing Transformation
(Perspective / Orthographic)

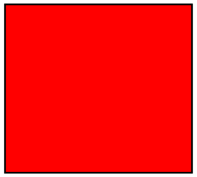
Clipping

Scan Conversion
(Rasterization)

Blending+Depth Test

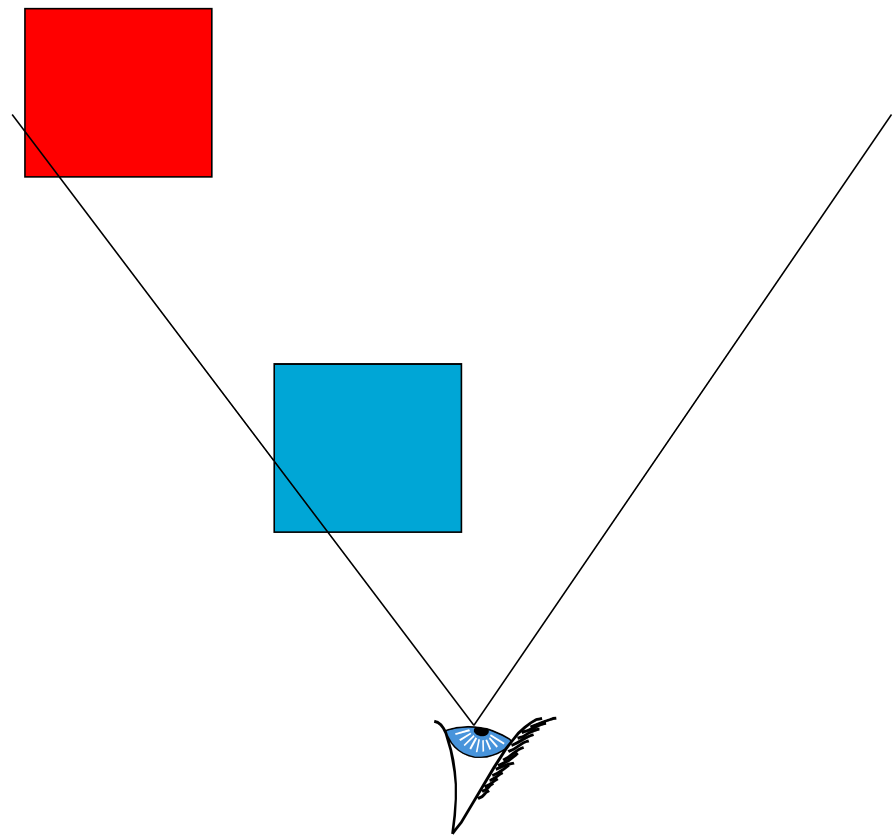
Graphics Pipeline

- Object Transformation
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Scan Conversion (Rasterization)
- Blending+Depth Test



Graphics Pipeline

- Object Transformation
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Scan Conversion (Rasterization)
- Blending+Depth Test



Graphics Pipeline

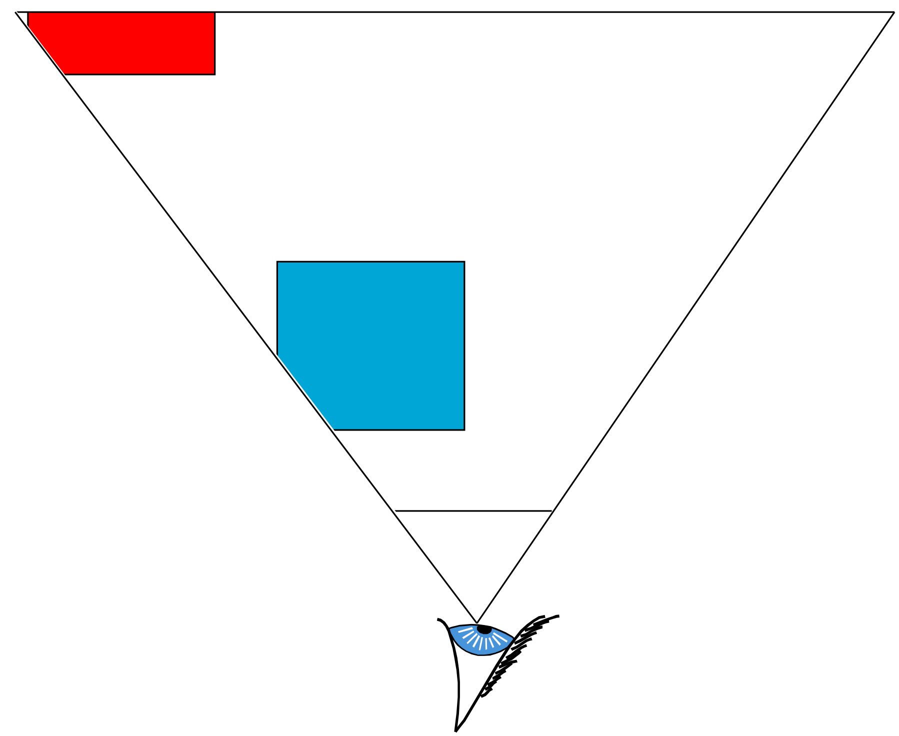
Object Transformation

Viewing Transformation (Perspective / Orthographic)

Clipping

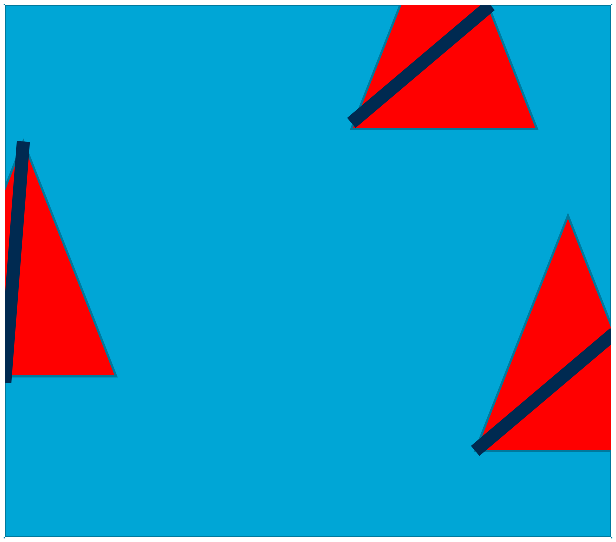
Scan Conversion (Rasterization)

Blending+Depth Test



Clip Space

- Create clipped triangles



Graphics Pipeline

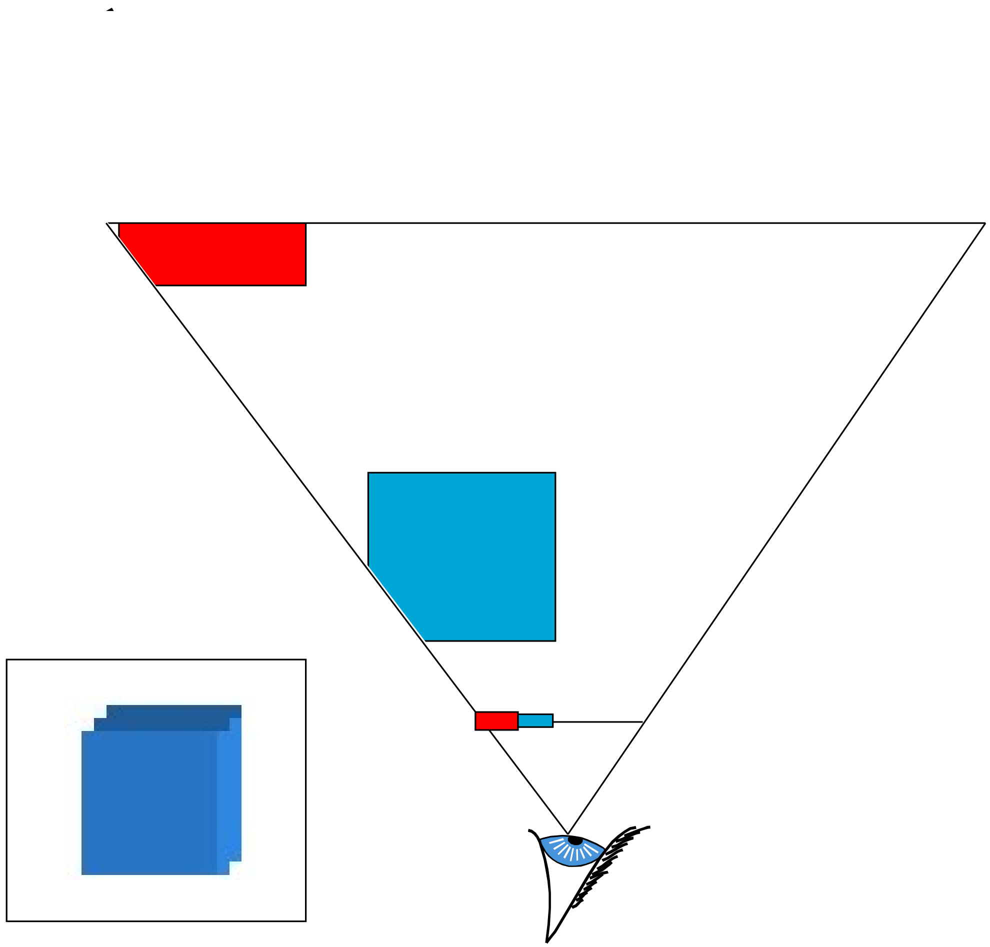
Object Transformation

Viewing Transformation
(Perspective / Orthographic)

Clipping

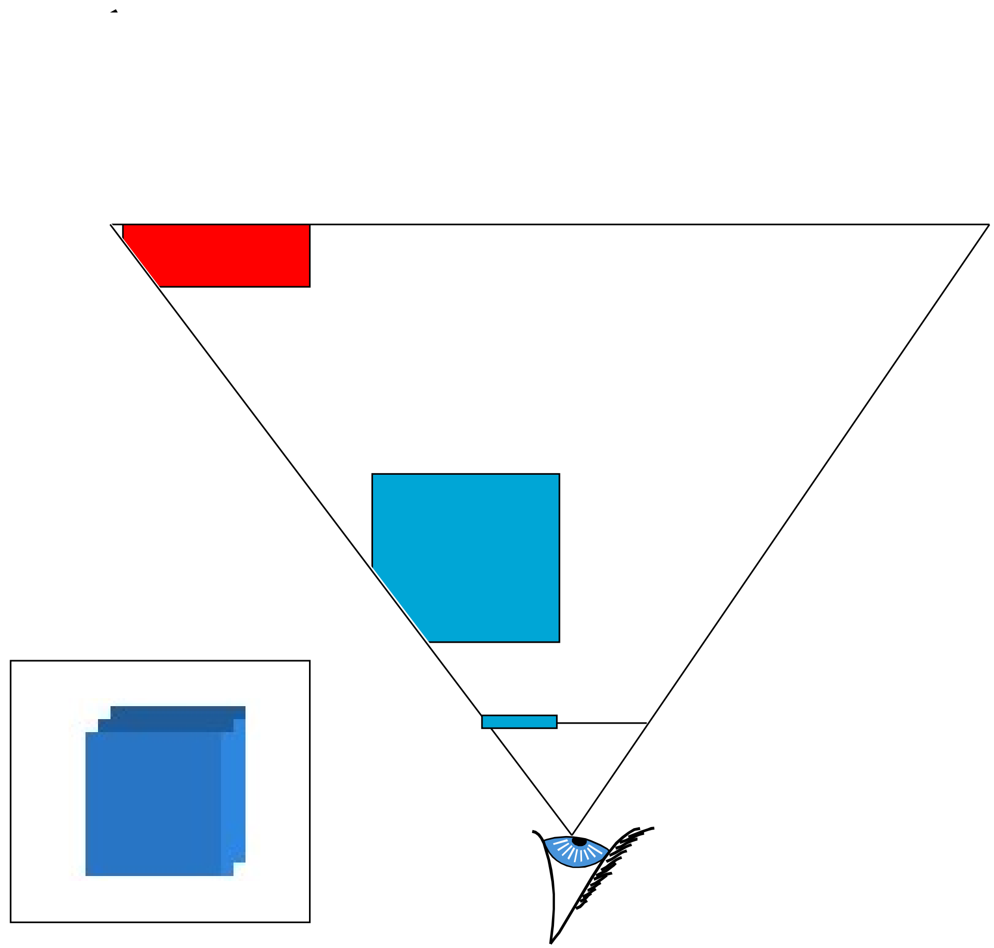
Scan Conversion
(Rasterization)

Blending+Depth Test

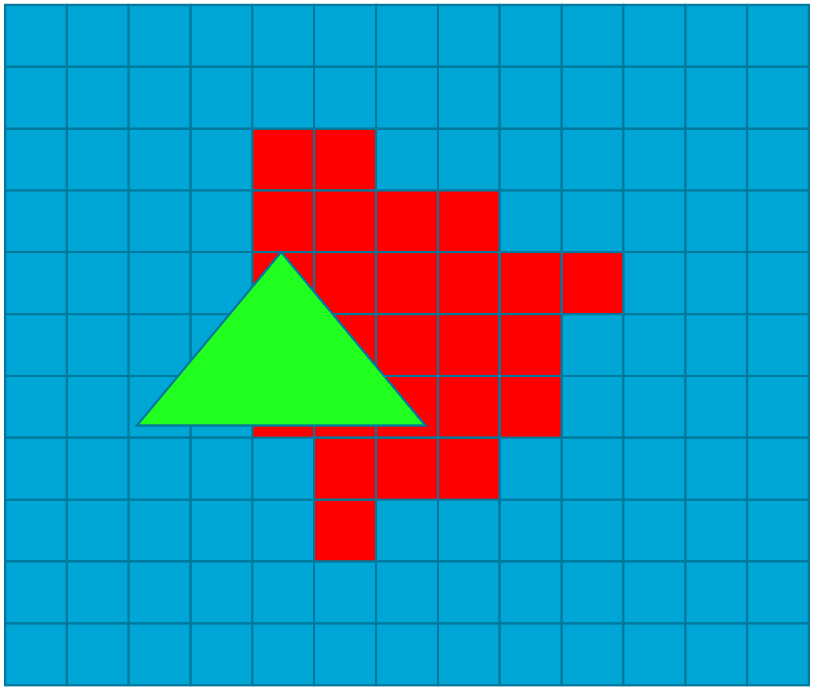


Graphics Pipeline

- Object Transformation
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Scan Conversion (Rasterization)
- Blending+Depth Test

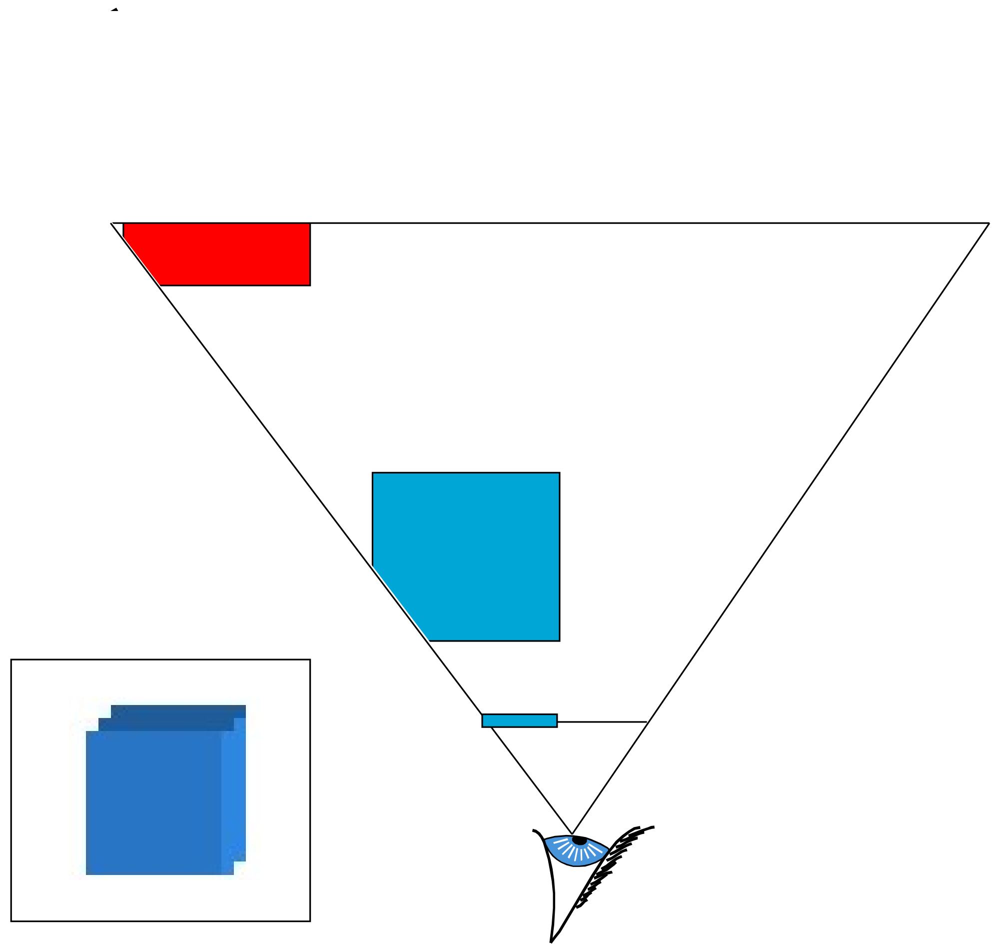


Blending and Depth Test



Graphics Pipeline

- Object Transformation
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Scan Conversion (Rasterization)
- Blending+Depth Test

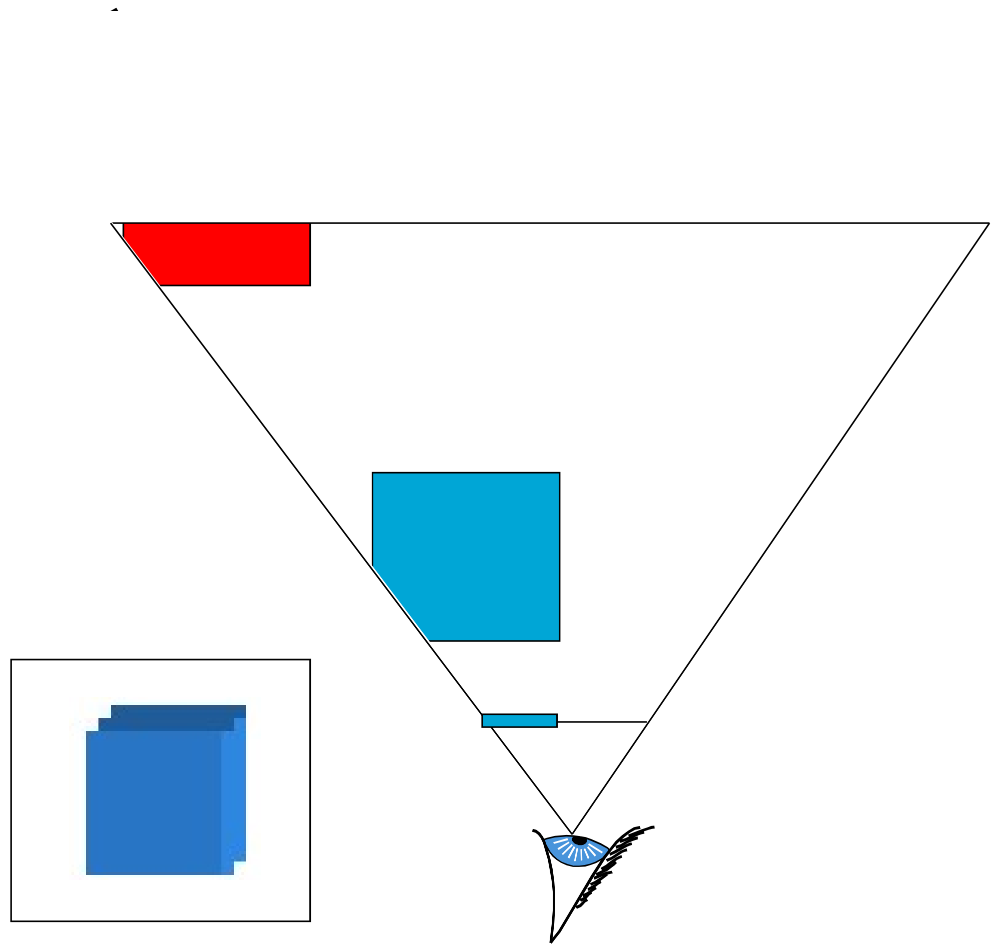


Questions?

Based on https://steemit.com/dmania/@m2nnari/cpu-memes-zg1hbmlh-2htsm

Standard Graphics Pipeline

- Object Transformation
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Scan Conversion (Rasterization)
- Blending+Depth Test



Today’s Modern/Programmable Graphics Pipeline

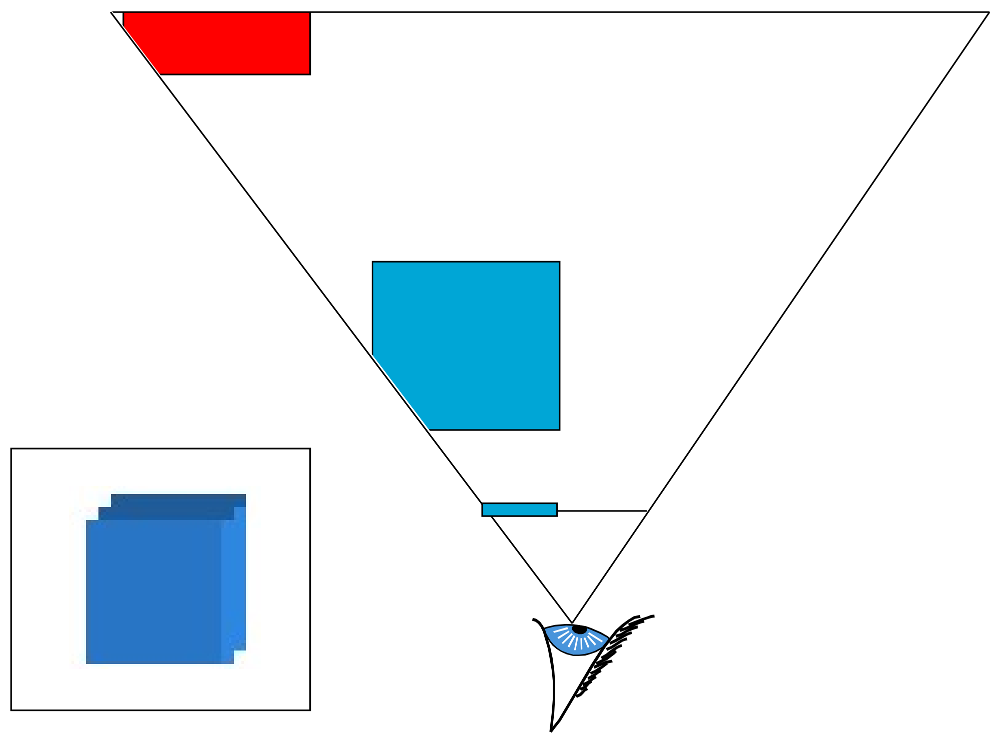
Vertex Shader

Geometry Shader

Rasterization

Fragment Shader

Blending+Depth Test



Programmable Graphics Pipeline

On today's GPUs stages are customizable (programmable)

- Vertex shader: Computations on vertices
- Geometry shader: Computations on primitives
  (primitives = triangles, quads...)
- Fragment shader: Computations on pixels
  (Fragment=pixel+attributes)

Graphics API

- To communicate and control the graphics card, we can make use of special APIs.
- The most famous APIs are
  - Direct X
  - OpenGL
  - Vulkan

- In this course, we will focus on OpenGL
- To program the shaders, we will use GLSL (OpenGL shading language)

What makes a GPU fast?

- Very high parallel throughput
- The data typically resides on the GPU
  and is streamed through specialized programs

- To illustrate parallel processing:
  imagine a stream of vertices,
  groups of vertices are processed in parallel
  (e.g., each vertex is multiplied by a matrix)
  result is put back into a stream

Flat Mesh Representation

- Simply store vertices in order

Vertex Positions: [x0,y0,z0, x1, y1, z1, x2,y2,z2,…xn, yn, zn]

T0

- What about attributes like color?

Colors: [r0,g0,b0, r1, g1, b1, c2,c2,c2,…cn, cn, cn]

Color of
first vertex

- What is suboptimal about this?

Optimized Mesh Representation

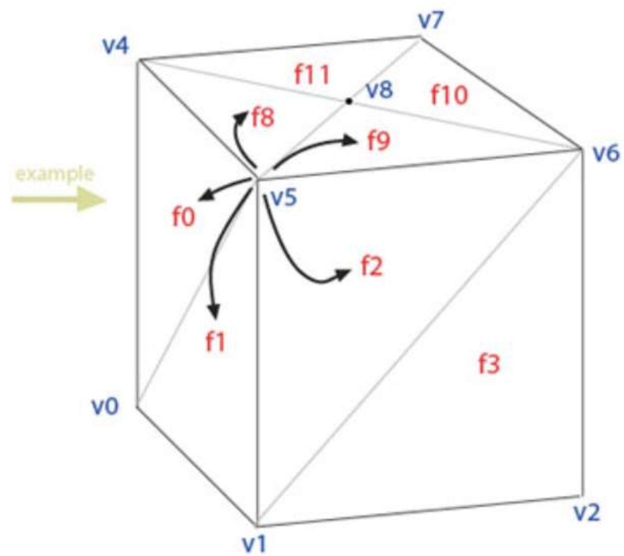
- Indexed Face Set

Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

v0	0,0,0
v1	1,0,0
v2	1,1,0
v3	0,1,0
v4	0,0,1
v5	1,0,1
v6	1,1,1
v7	0,1,1
v8	.5,.5,0
v9	.5,.5,1



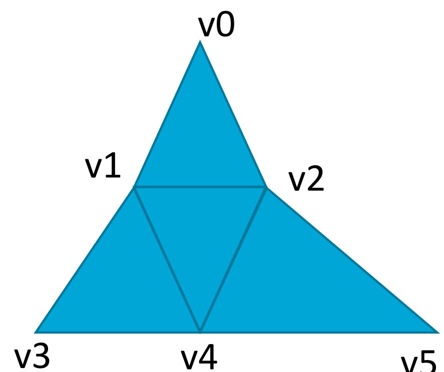
Mesh with attributes GPU-ready

- Data stored in arrays
  containing all vertices **and attributes**
- Triangles are defined by an index array:
  consecutive indices in this array define faces
- Advantage: Shared vertices only stored once

Vertex Positions: [x0,y0,z0, x1, y1, z1, …x5, y5, z5]
Attributes (can be several): [r0,g0,b0, r1, g1, b1, … r5, g5, b5]
IndexBuffer: [0,2,1,1,3,4,1,4,2,2,4,5]

T0 T1 T2 T3

e.g., T0 uses indices 0, 2, 1 meaning it uses vertices v0, v2, and v1.
Given an index, we can retrieve the data of the vertex from
the other arrays. For v0: x0, y0, z0 and r0, g0, b0



How often would all attributes of v1 be stored if not using this representation?

Today's Standard Graphics Pipeline

Vertex Shader

Geometry Shader

Rasterization

Fragment Shader

Blending+Depth Test

Today’s Standard Graphics Pipeline

Vertex Shader

Geometry Shader

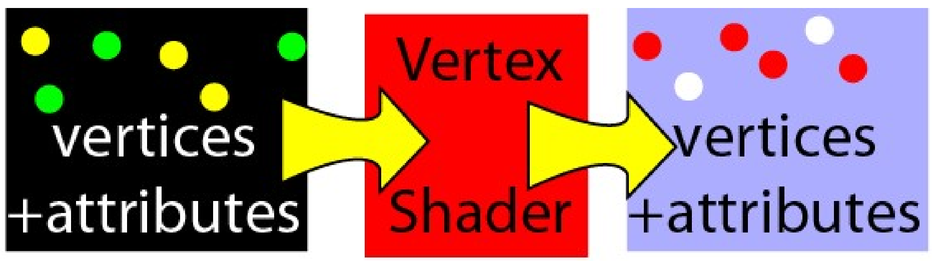
Rasterization

Fragment Shader

Blending+Depth Test

Vertex Shader

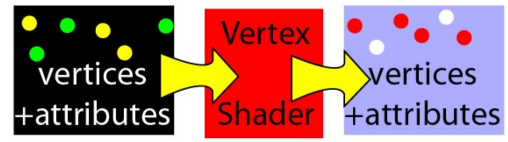
- Input: Vertex with attributes
- Output: Vertex with attributes



- What operation is done in standard pipeline?

Usually camera projection / object placement!

Vertex Shader Example (GLSL 4.3)



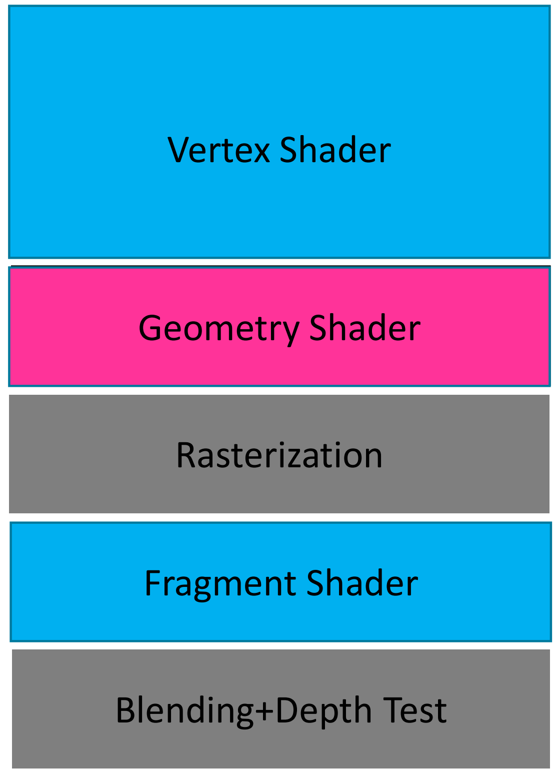
Constant for all vertices

```
#version 430

layout(location=0) uniform mat4 ModelViewMatrixValue;

layout(location=0) in vec4 pos; //world-space position

layout(location=1) in vec3 normal; //world-space normal


//Data to be passed to geometry shader

out vec3 gColor;

void main(){

  gl_Position = ModelViewMatrixValue * pos;

  gColor = vec3(normal.x,0,0); //normal.x used as color

}
```

Example applications?

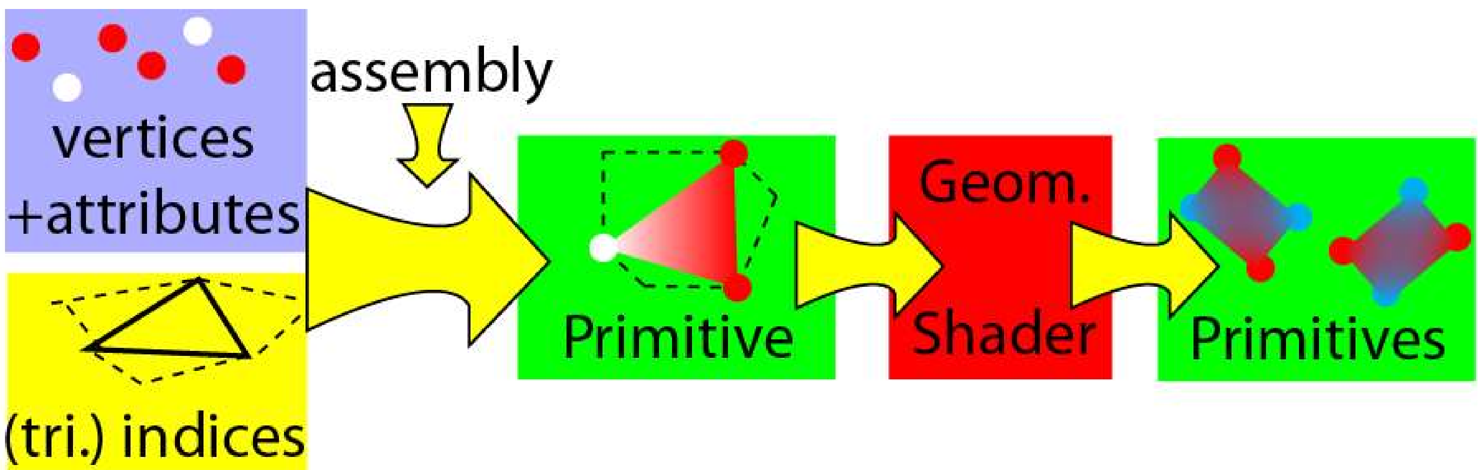
TUDelft

Today's Standard Graphics Pipeline



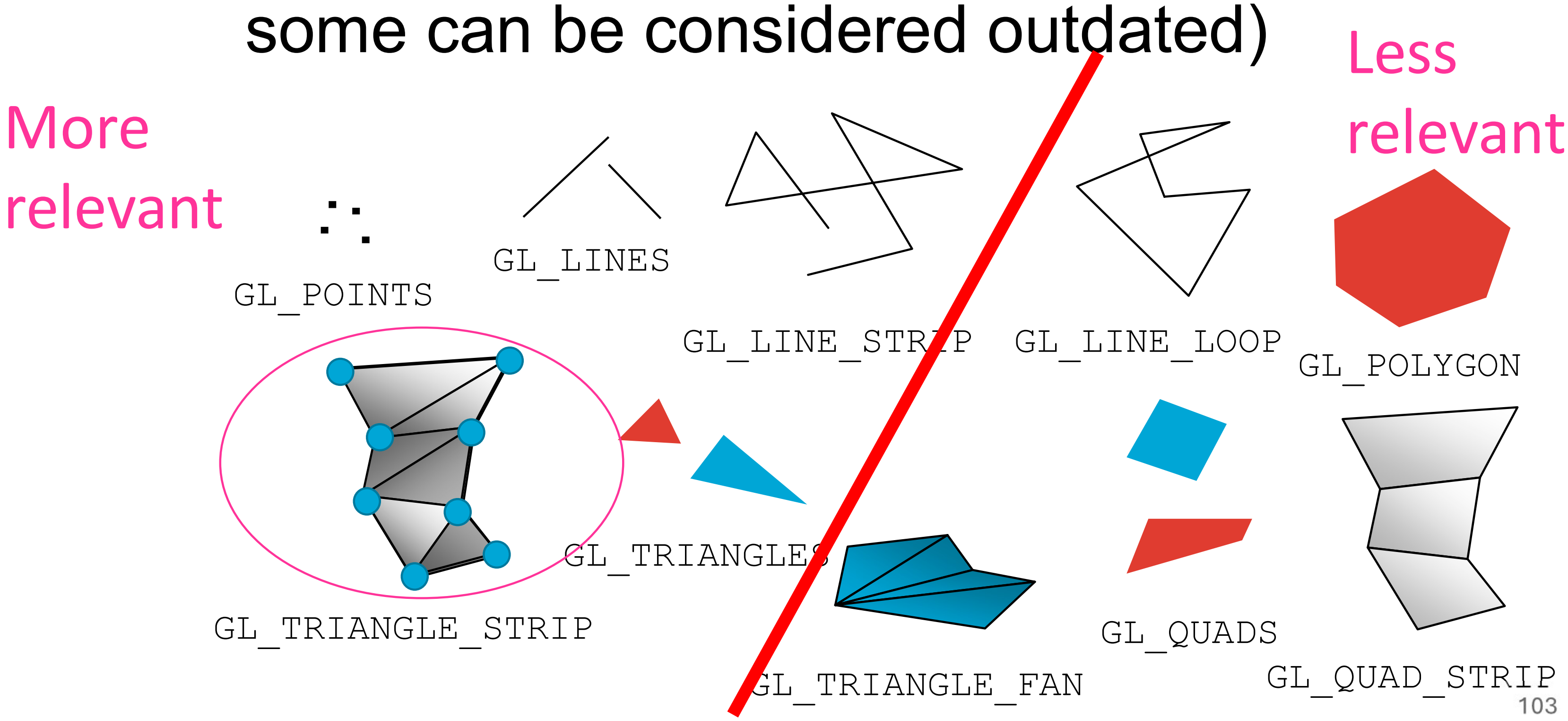
Geometry Shader

- Input: Vertex/Attribute **array** of current primitive (e.g., triangle or its immediate neighborhood)
- Output: Several primitives (vertices + attributes)



Primitives?

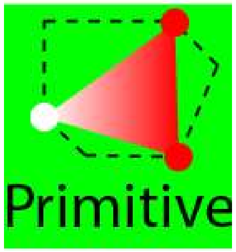
- Primitives are specified by vertices
  (These are the traditional OpenGL primitives – some can be considered outdated)

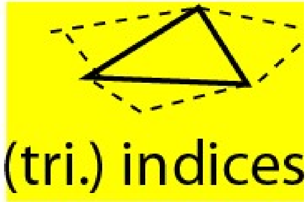


Geometry Shader: Input

- Geometry shader interprets input as:
  Points, lines, lines with adjacency, triangles or triangles with adjacency

- Default usage: current triangle's vertices are accessible
  - Works like for the indexed face set, with indices 0,1,2

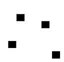




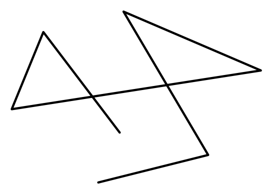
- Per extension: triangles/lines + neighbors
  (this requires special data formatting, just using strips does not work)

Geometry Shader: Output

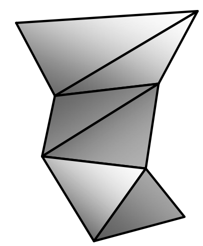
- Geometry shader supports **points, line_strip and triangle_strip** as output



GL_POINTS



GL_LINE_STRIP



GL_TRIANGLE_STRIP

Geometry Shader Example


```
#version 430
layout (triangles) in;
layout (triangle_strips, max_vertices=3) out;
in vec3 gColor[3];
out vec3 fColor;
void main(void){
      for (int i=0;i<3;++i) {
            gl_Position=gl_in[i].gl_position;
            fColor=gColor[i];
            EmitVertex();
      }
      EndPrimitive();
}
```

Geometry Shader Example

```
#version 430
layout (triangles) in;
layout (triangle_strips, max_vertices=3) out;
in vec3 gColor[3];
out vec3 fColor;
void main(void){
      for (int i=0;i<3;++i) {
            gl_Position=gl_in[i].gl_position;
            fColor=gColor[i];
            EmitVertex();
      }
      EndPrimitive();
}
```

Arrays coming from vertex shader containing the data of the current triangle

Geometry Shader Example

```
#version 430
layout (triangles) in;
layout (triangle_strips, max_vertices=3) out;
in vec3 gColor[3];
```

Arrays coming from vertex shader containing the data of the current triangle

```
out vec3 fColor;
void main(void){
        for (int i=0;i<3;++i) {
                gl_Position=gl_in[i].gl_position;
                fColor=gColor[i];
```

Use current primitive's data (position and color) and store them

```
                EmitVertex();
```

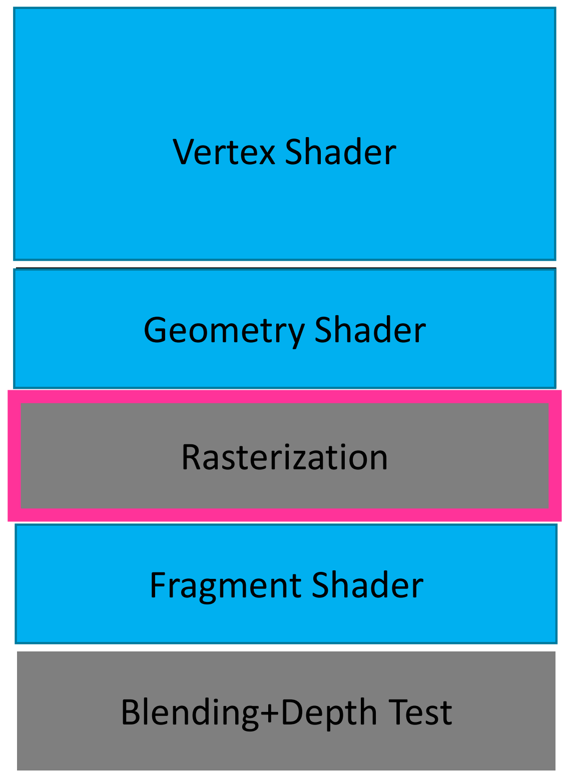
Create a vertex

```
        }
        EndPrimitive();
```

Build a primitive (here a triangle) out of the created vertices

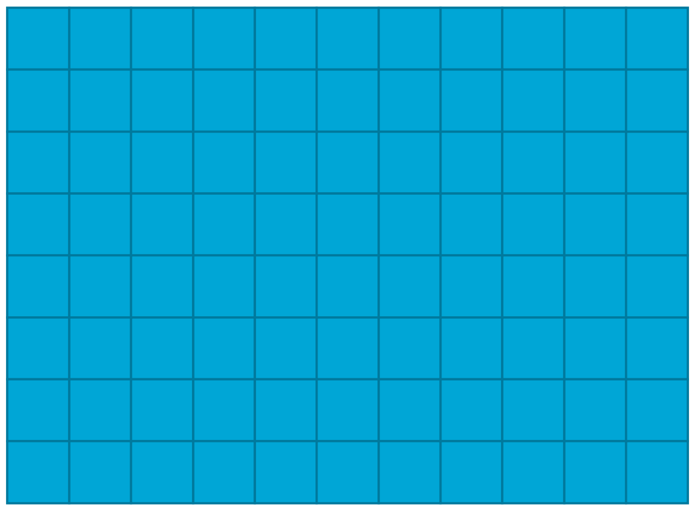
```
}
```

Today’s Standard Graphics Pipeline



Rasterization

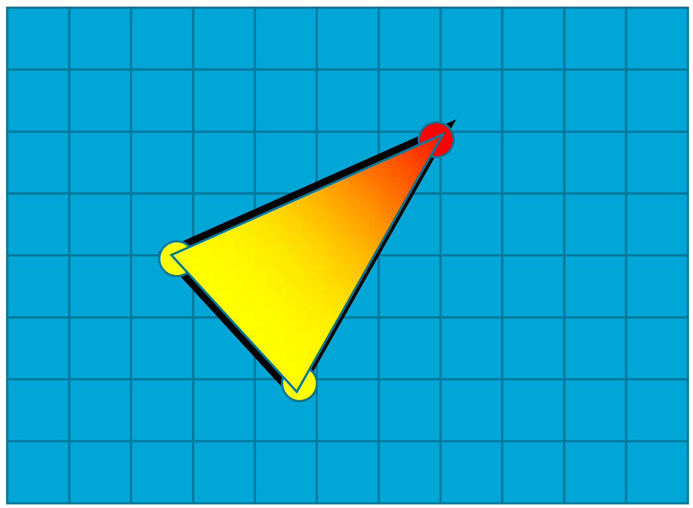
- Triangles into fragments (pixels+attributes)



- Fragment data determined by interpolation
- Values are extracted at pixel centers

Rasterization

- Triangles into fragments (pixels+attributes)



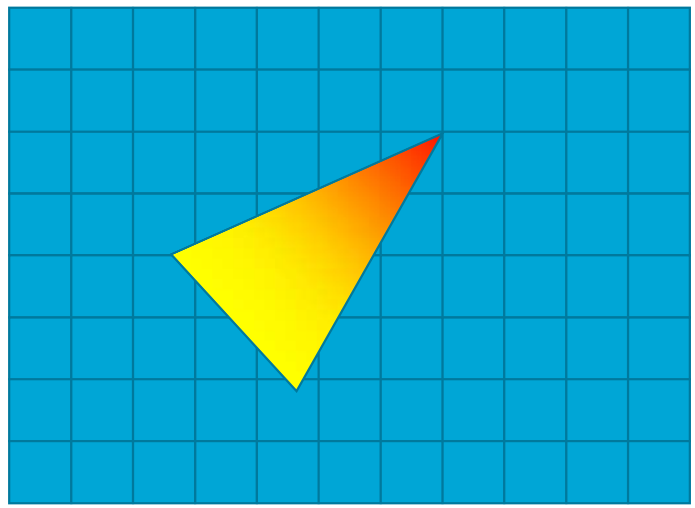
Two yellow,
one red vertex

Colors are
interpolated over
triangle

- Fragment data determined by interpolation
- Values are extracted at pixel centers

Rasterization

- Triangles into fragments (pixels+attributes)



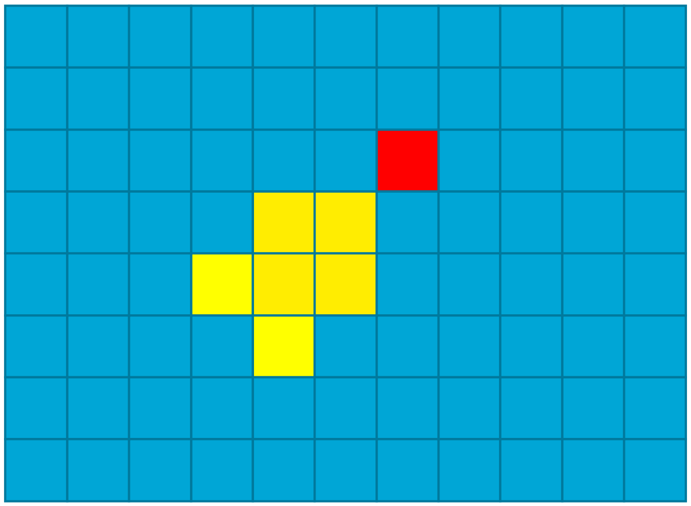
Two yellow,
one red vertex

Colors are
interpolated over
triangle

- Fragment data determined by interpolation
- Values are extracted at pixel centers

Rasterization

- Triangles into fragments (pixels+attributes)

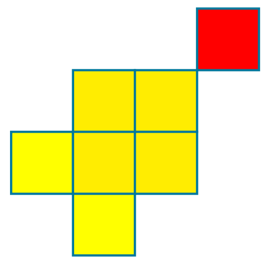


Colors sampled
at pixel centers

- Fragment data determined by interpolation
- Values are extracted at pixel centers

Rasterization

- Triangles into fragments (pixels+attributes)



The input to the fragment shader

- Fragment data determined by interpolation
- Values are extracted at pixel centers

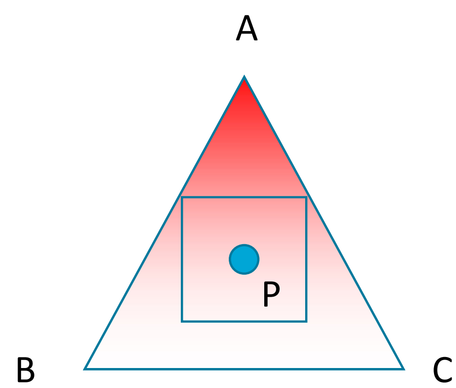
Example: Interpolation

- What are "interpolated" vertex attributes in each Fragment?

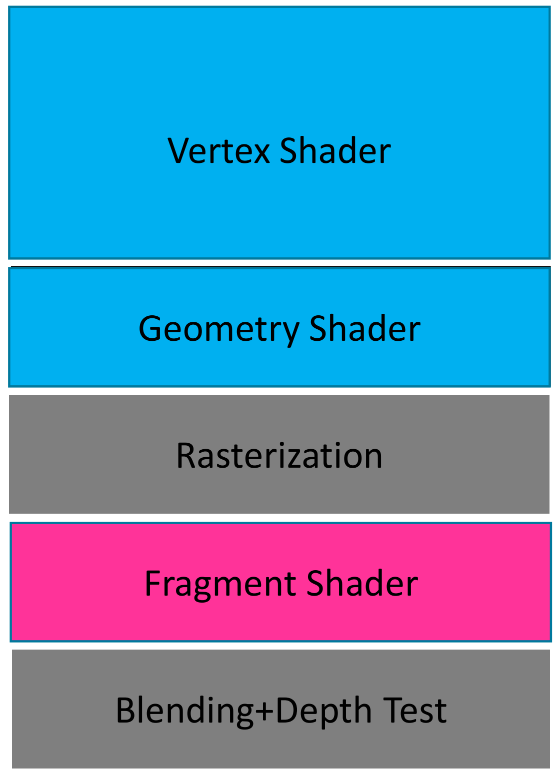
- Imagine P=1/3 A + 1/3 B + 1/3 C
- If the colors at A, B, C are red (1,0,0), white (1,1,1), white (1,1,1) respectively

- The interpolated color at P would be:

  (1, 2/3, 2/3), which is pink

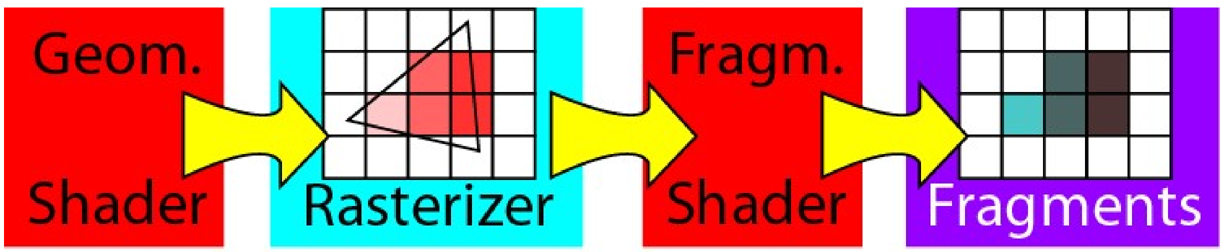


Today’s Standard Graphics Pipeline



Fragment Shader

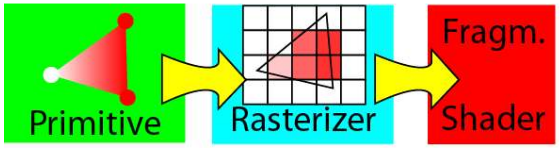
- Input: “Pixel” with interpolated vertex/attributes
- Output: New fragment for this position
  - Typically a new pixel color is computed



Fragment Shader

- Fragment = colors + depth at a pixel location
- Two fragments can fall in the same pixel
  (e.g., two overlapping triangles)
- Fragments **cannot** be moved in the Fragment shader,
  their location on the screen is fixed

Fragment Shader Example



Vertex values are interpolated!!!

```
#version 430
in vec3 fColor;
layout(location=0) out vec4 outColor;


void main()
{
	outColor = vec4(fColor,1.0);
}
```

Interpolated attributes (was specified on primitive vertices in the geometry shader)

Today’s Standard Graphics Pipeline

Vertex Shader

Geometry Shader

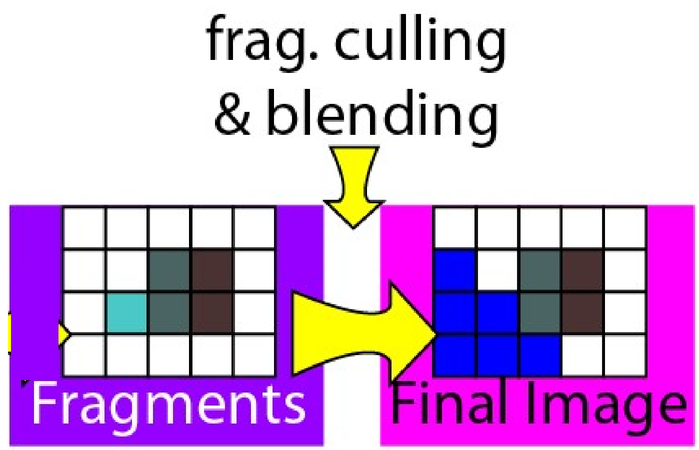
Rasterization

Fragment Shader

Blending+Depth Test

Blending Stage

- Blending stage cannot be programmed.
- It takes the produced fragment and combines it with the current image.



Blending Stage

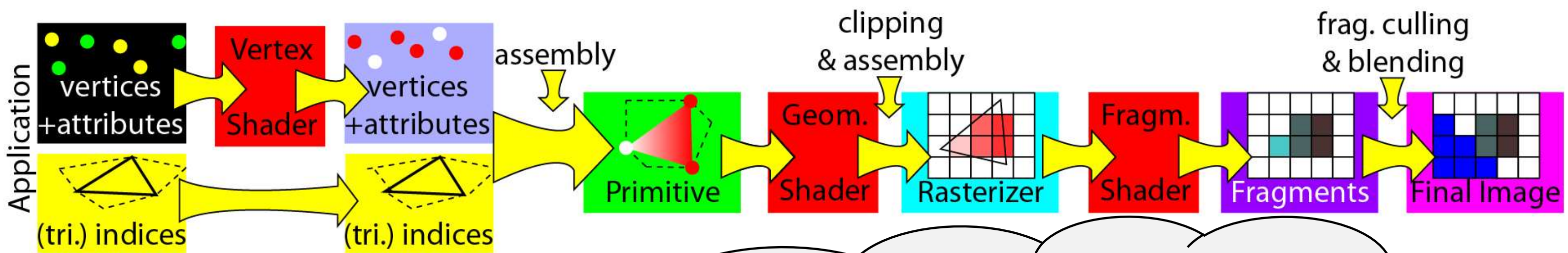
- Typically two choices exist:

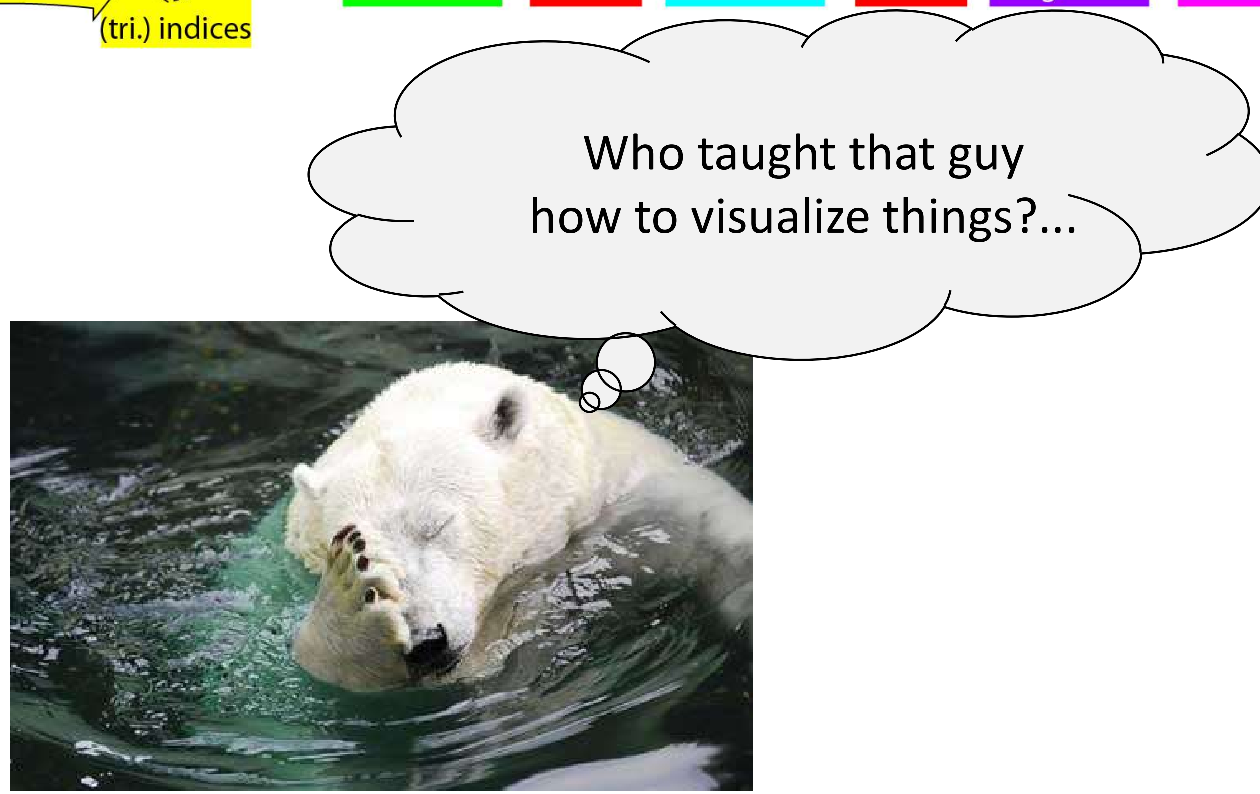
- Culling:
  Do not produce an output
  e.g., depth test failed or fragment is discarded in the code

- Blending:
  Combine background and new fragment
  e.g., sum current and incoming value
  (there are a few operations to choose from)

To be continued…

Programmable Graphics Pipeline



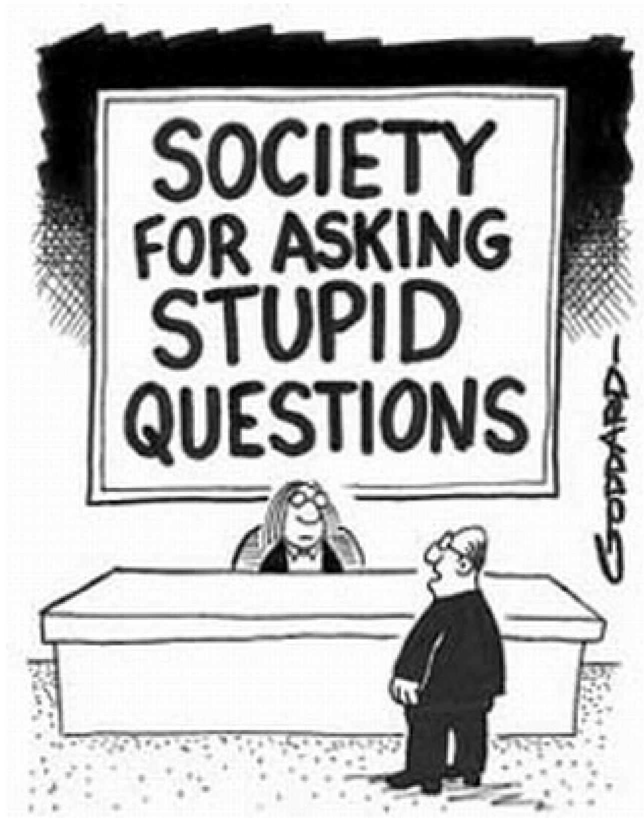


Cool stuff in the fragment shader!

- Similar means as in my demo:
- https://www.shadertoy.com/view/3lyBDw

- Using textures (a little outlook):
- https://www.shadertoy.com/view/XcXXzS
- https://www.shadertoy.com/view/XltGRX

Questions?



"Excuse me, is this the Society for Asking Stupid Questions?"

Thank you very much for your attention!