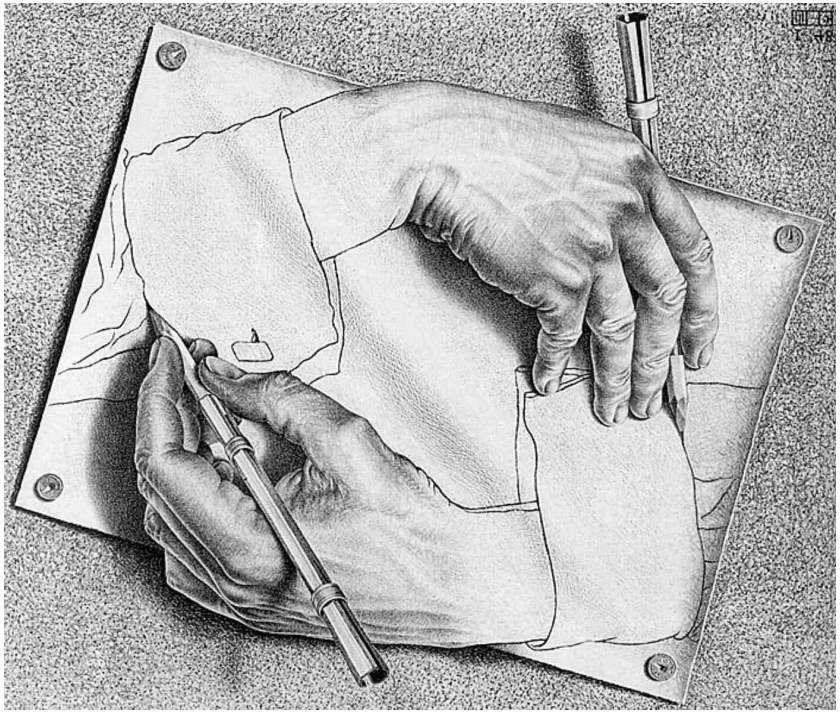
Game Technology

Lecture 3 – 7.11.2017
Software Rendering

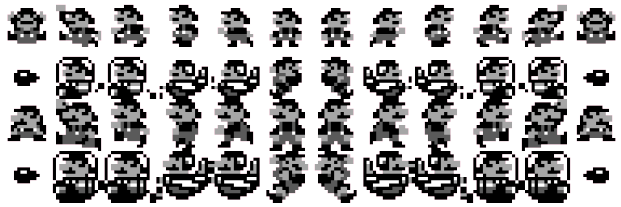
TECHNISCHE UNIVERSITÄT DARMSTADT



Dipl.-Inform. Robert Konrad
Polona Caserman, M.Sc.

Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

2D Rendering



Super Mario Land, 1989

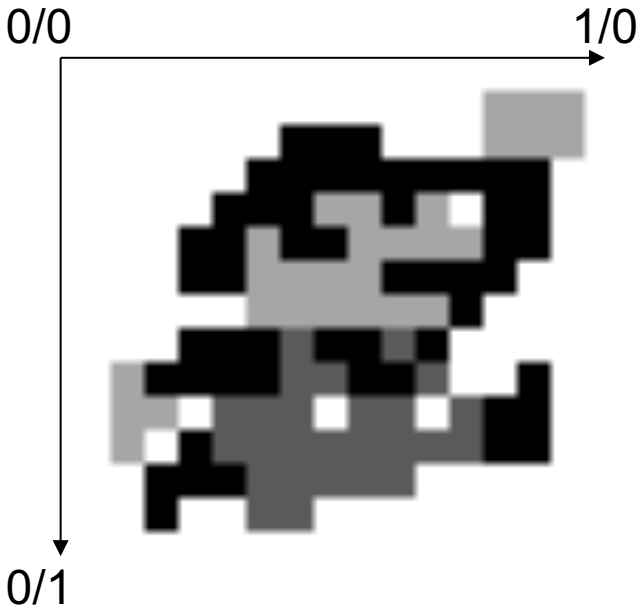
Positioning

TECHNISCHE UNIVERSITÄT DARMSTADT

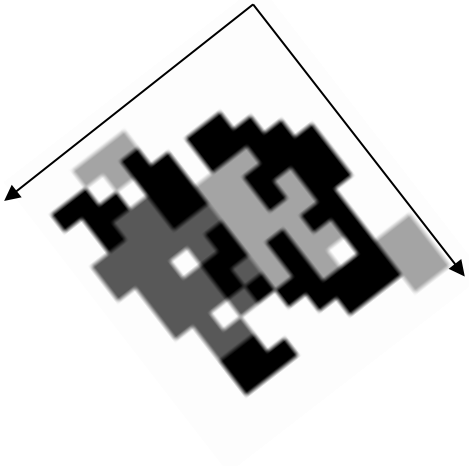
Nice property: No need to do heavy calculations on input pixels

Can „blit“ the image from one memory region to the screen buffer

Scaling



Rotations



Shearing

TECHNISCHE UNIVERSITÄT DARMSTADT

Fonts

TECHNISCHE UNIVERSITÄT DARMSTADT

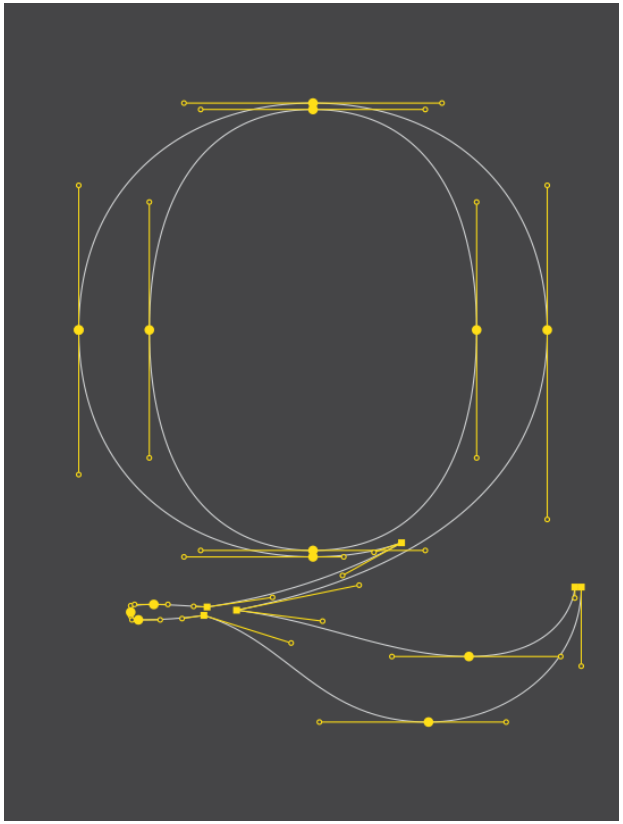
TrueType format

- Line segments and Bézier curves
- Kerning
  - VA
- Pixel snapping

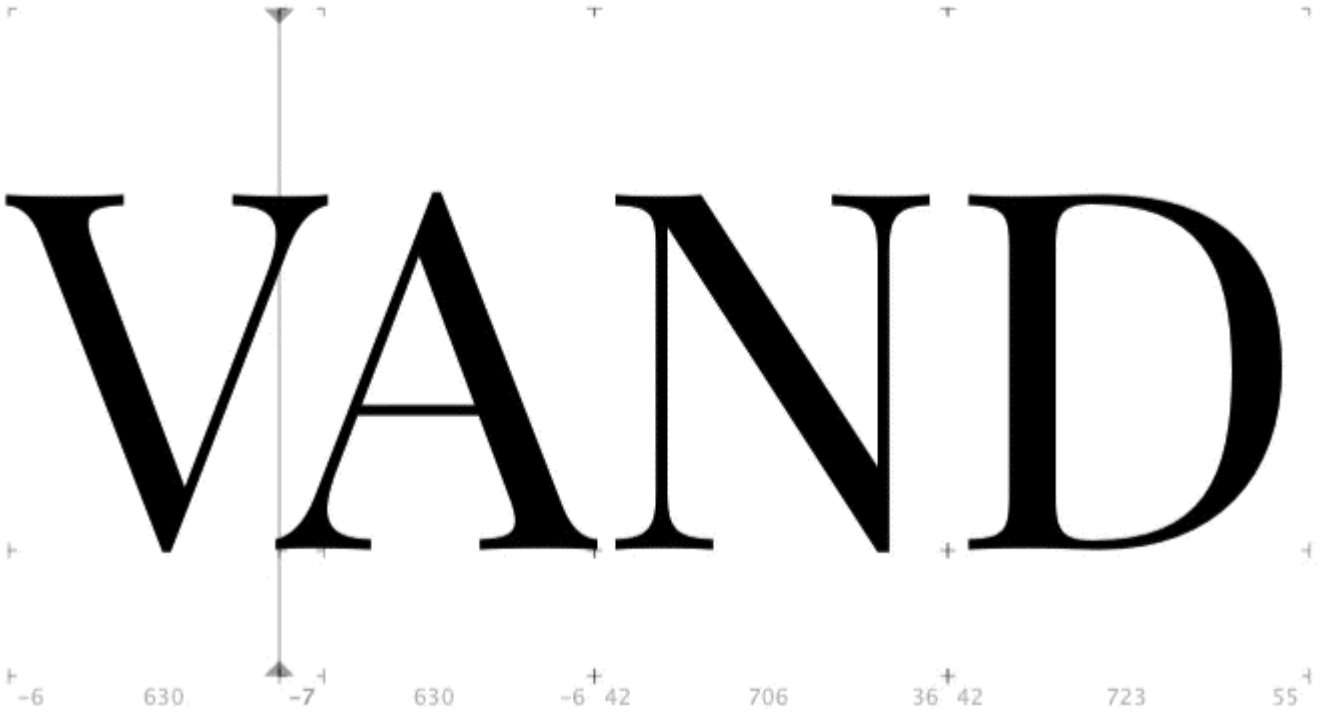
„TrueType systems include a virtual machine that executes programs inside the font”

Unicode

- More than 110,000 characters



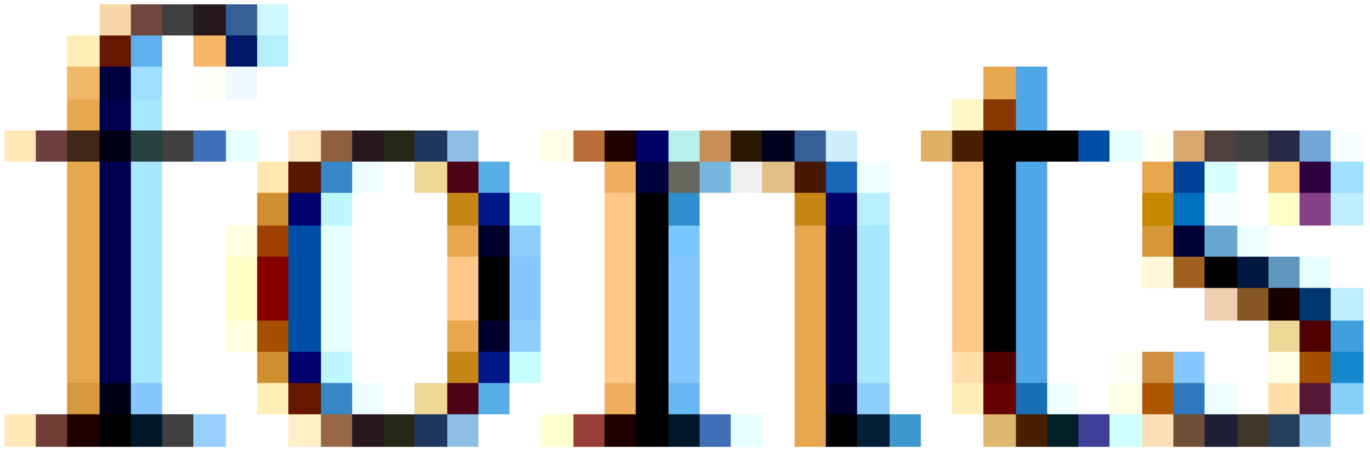
„VA“



Source: https://glyphsapp.com/tutorials/kerning

Subpixel Rendering

TECHNISCHE UNIVERSITÄT DARMSTADT



Bitmap Fonts

Font Libs

TECHNISCHE UNIVERSITÄT DARMSTADT

Freetype

http://www.freetype.org

stb_truetype

https://github.com/nothings/stb

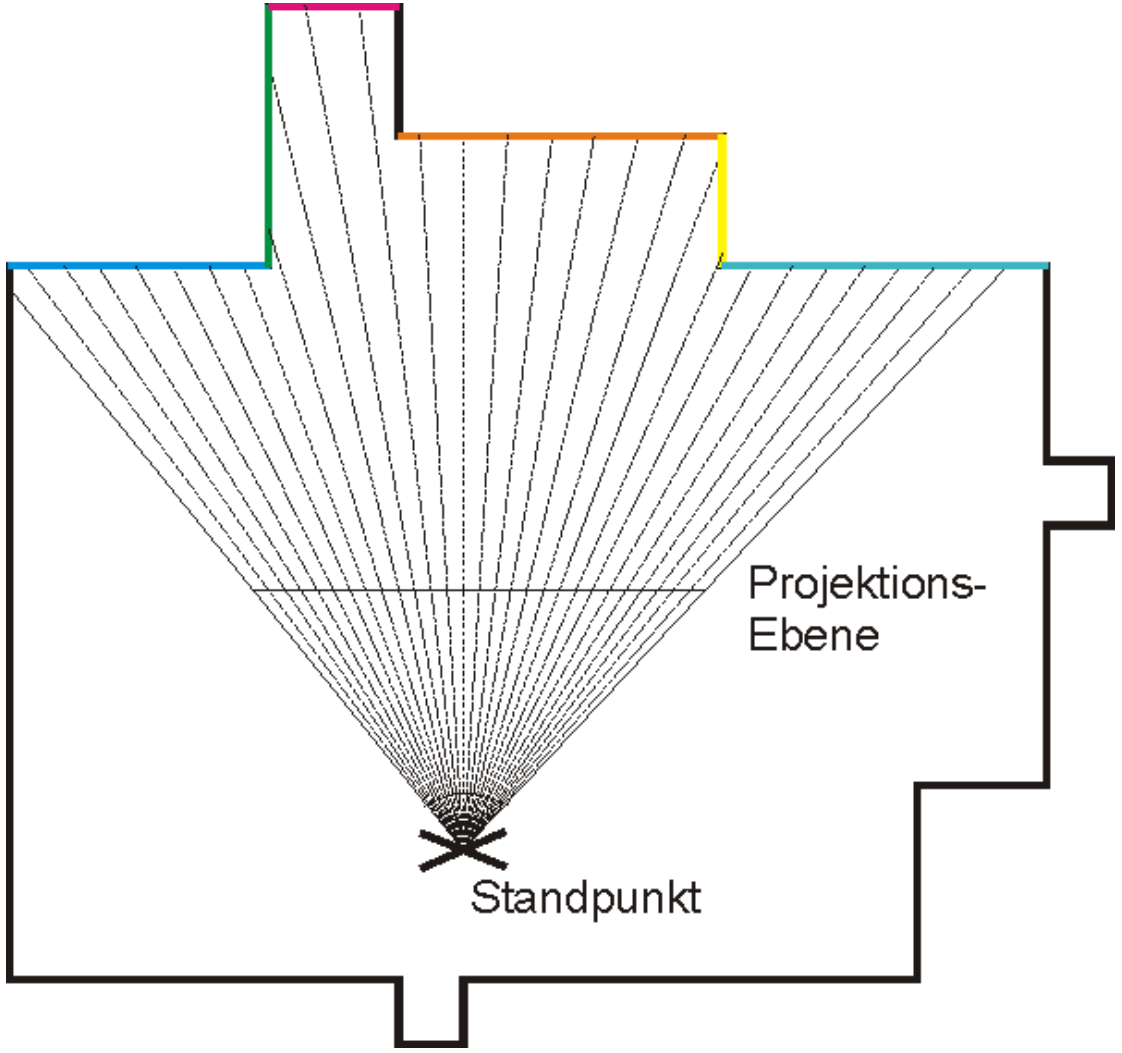
„Raycasting“

TECHNISCHE UNIVERSITÄT DARMSTADT

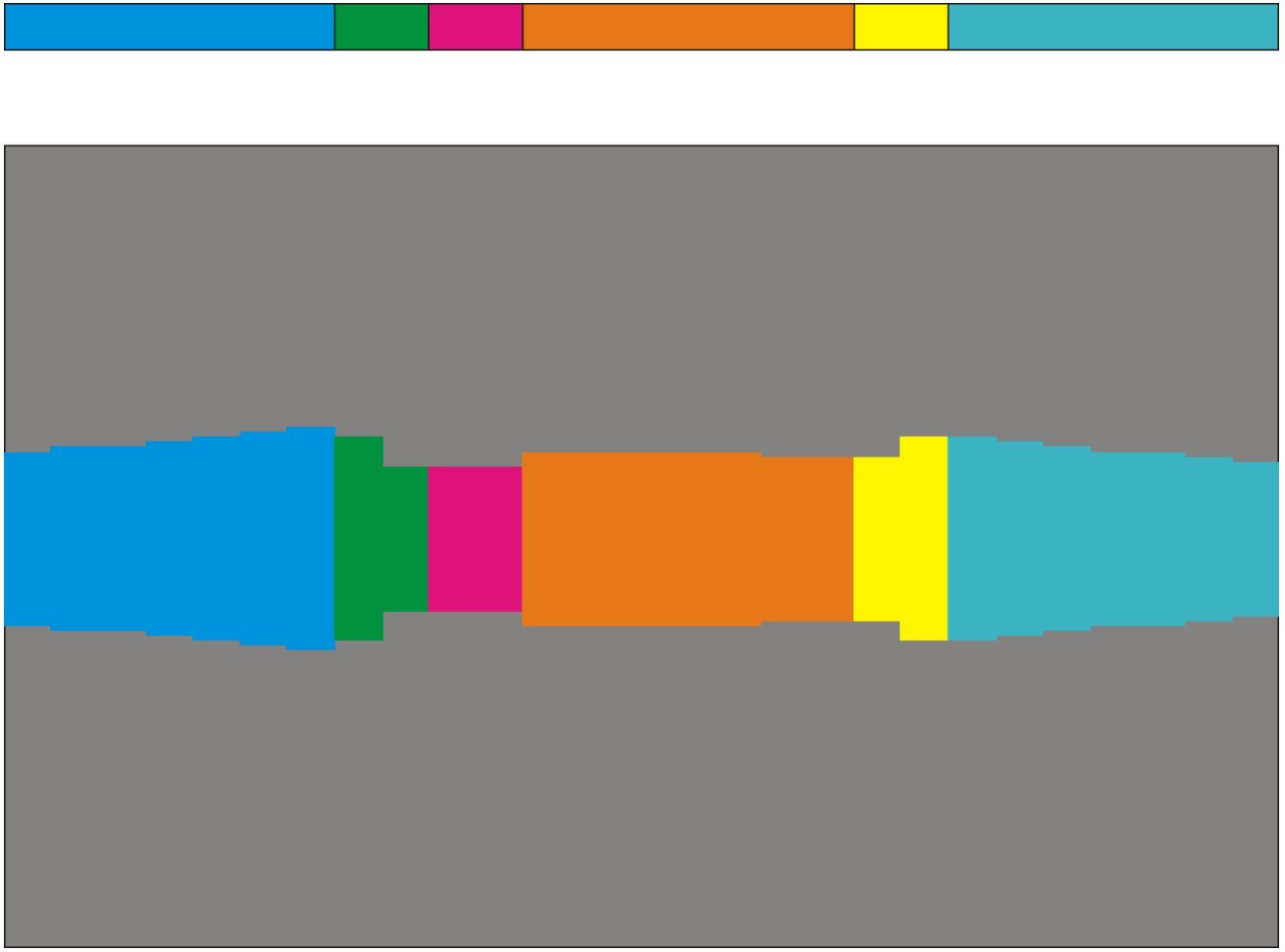
Catacomb 3-D, 1991

„Raycasting“

TECHNISCHE UNIVERSITÄT DARMSTADT

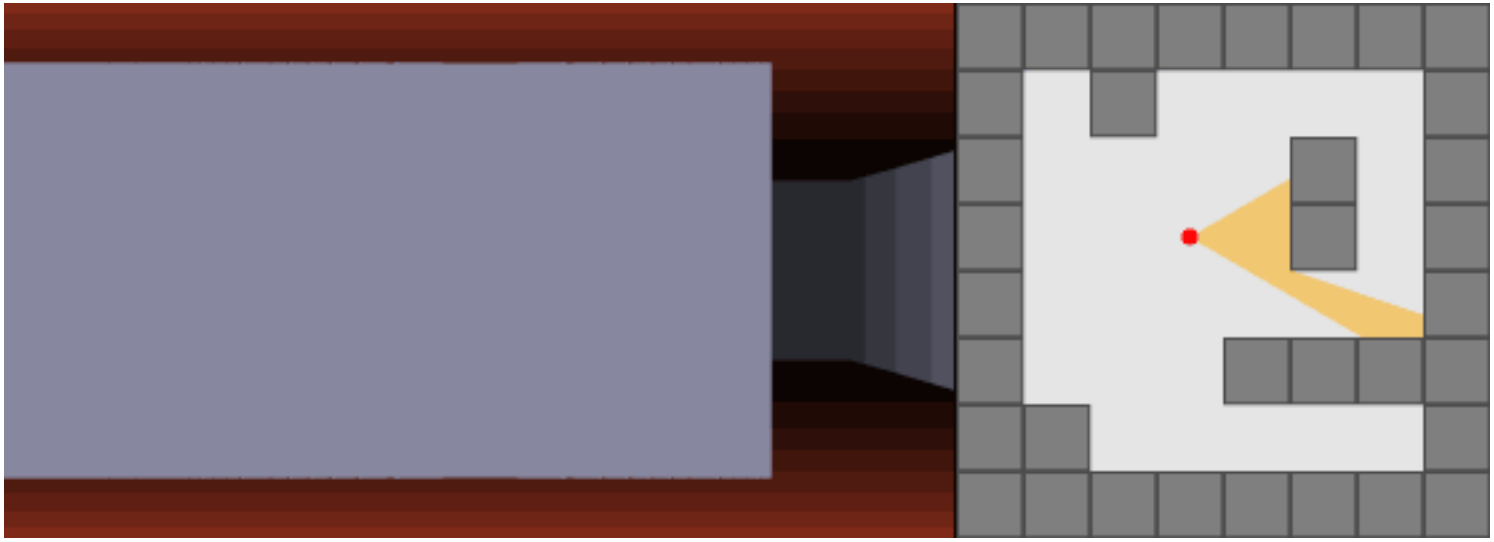


„Raycasting“



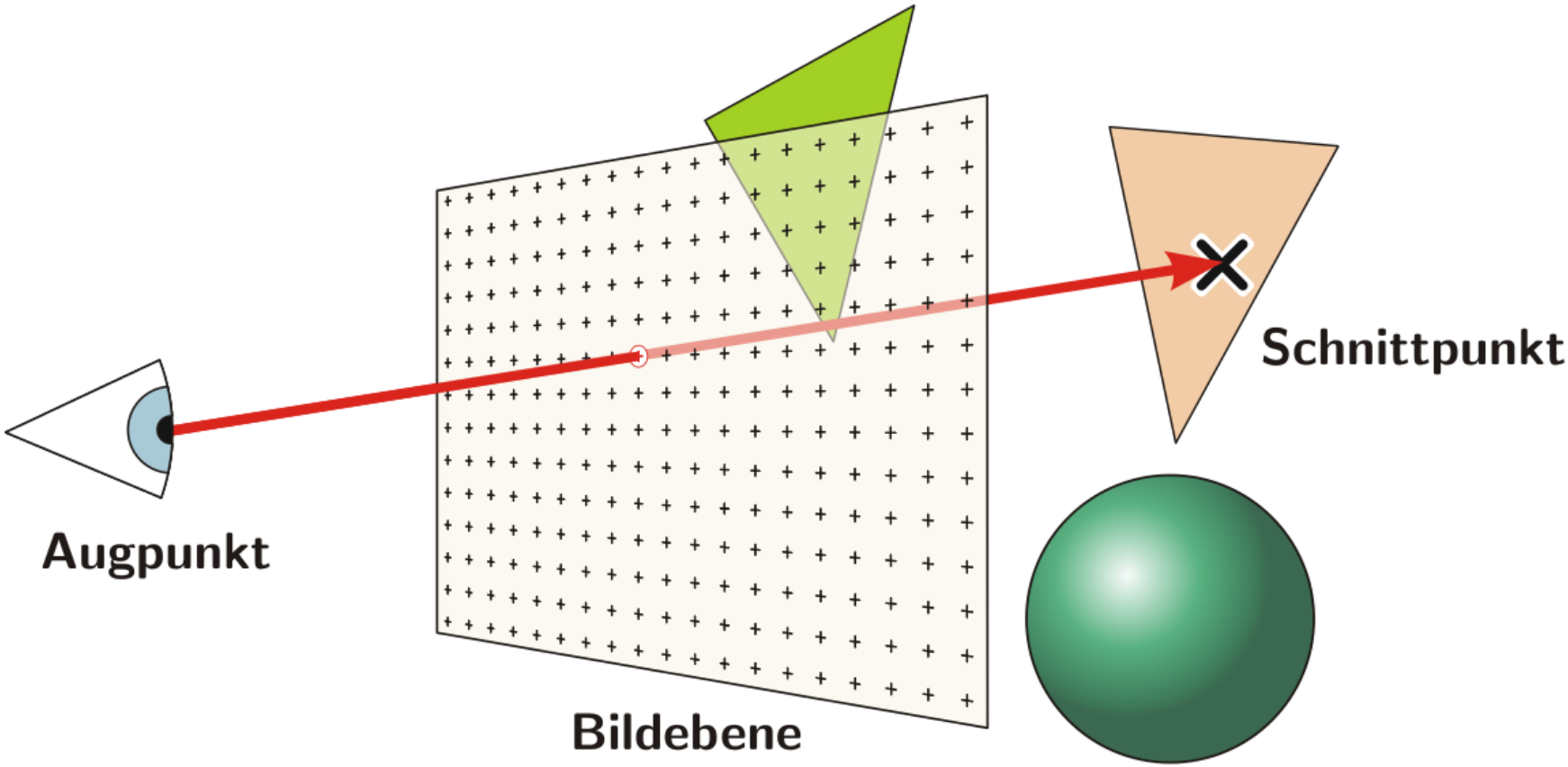
Source: https://de.wikipedia.org/wiki/Raycasting

„Raycasting“



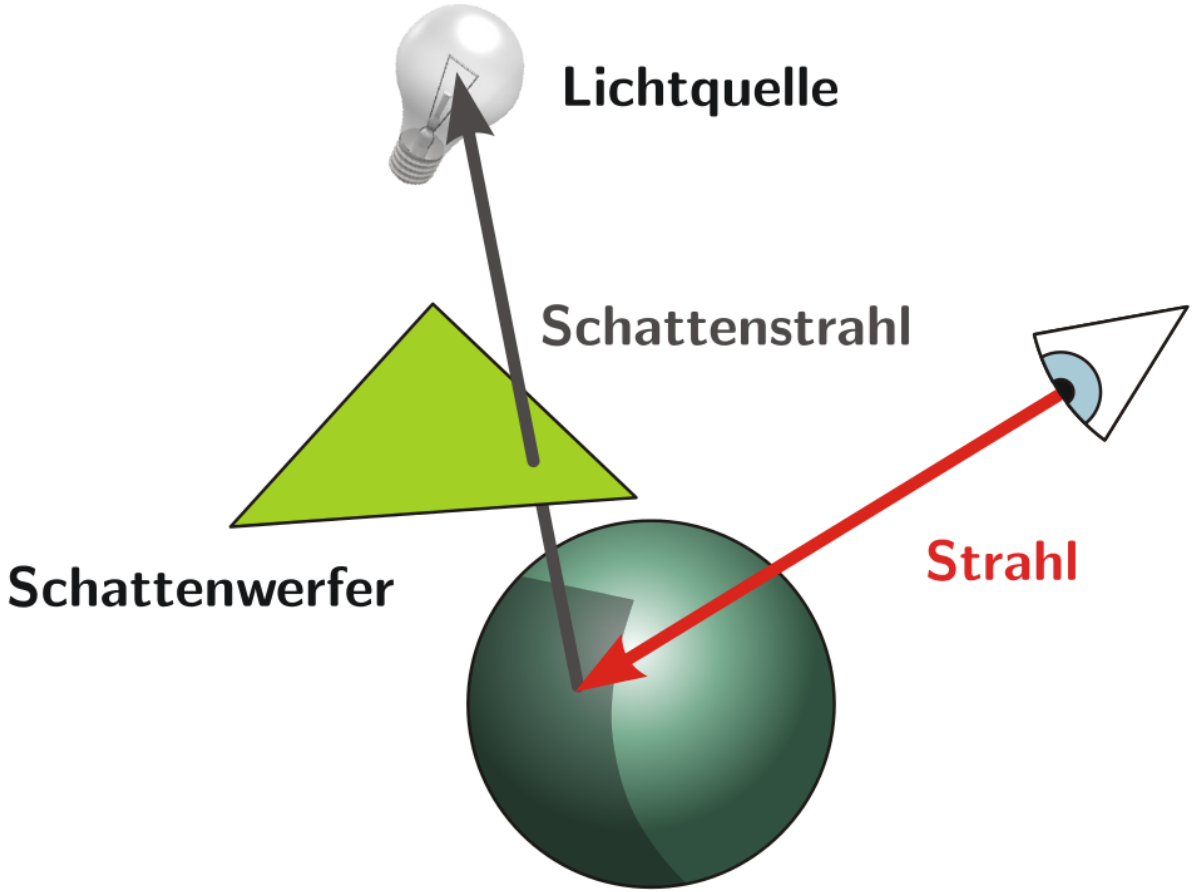
https://en.wikipedia.org/wiki/Wolfenstein_3D_engine

Raytracing

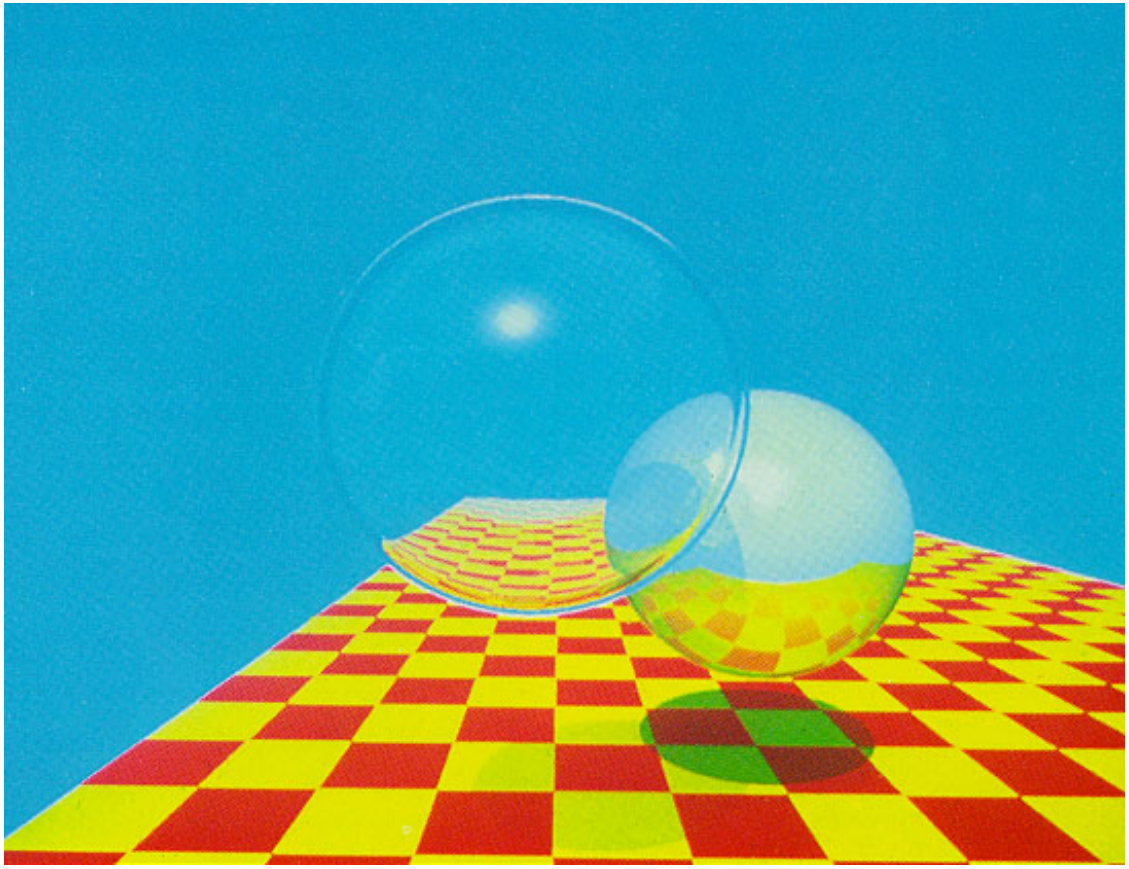


Source: https://de.wikipedia.org/wiki/Raytracing

Raytracing

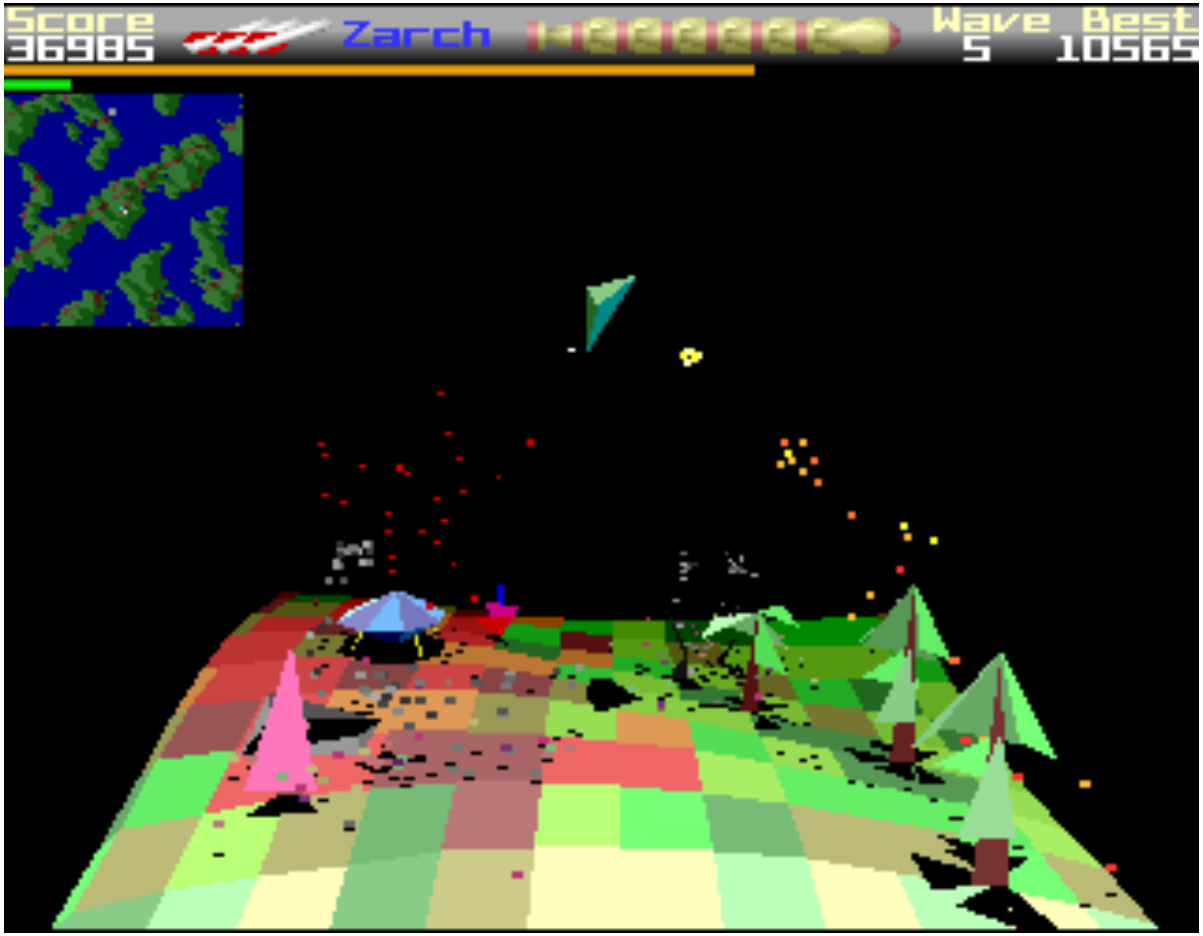


Raytracing



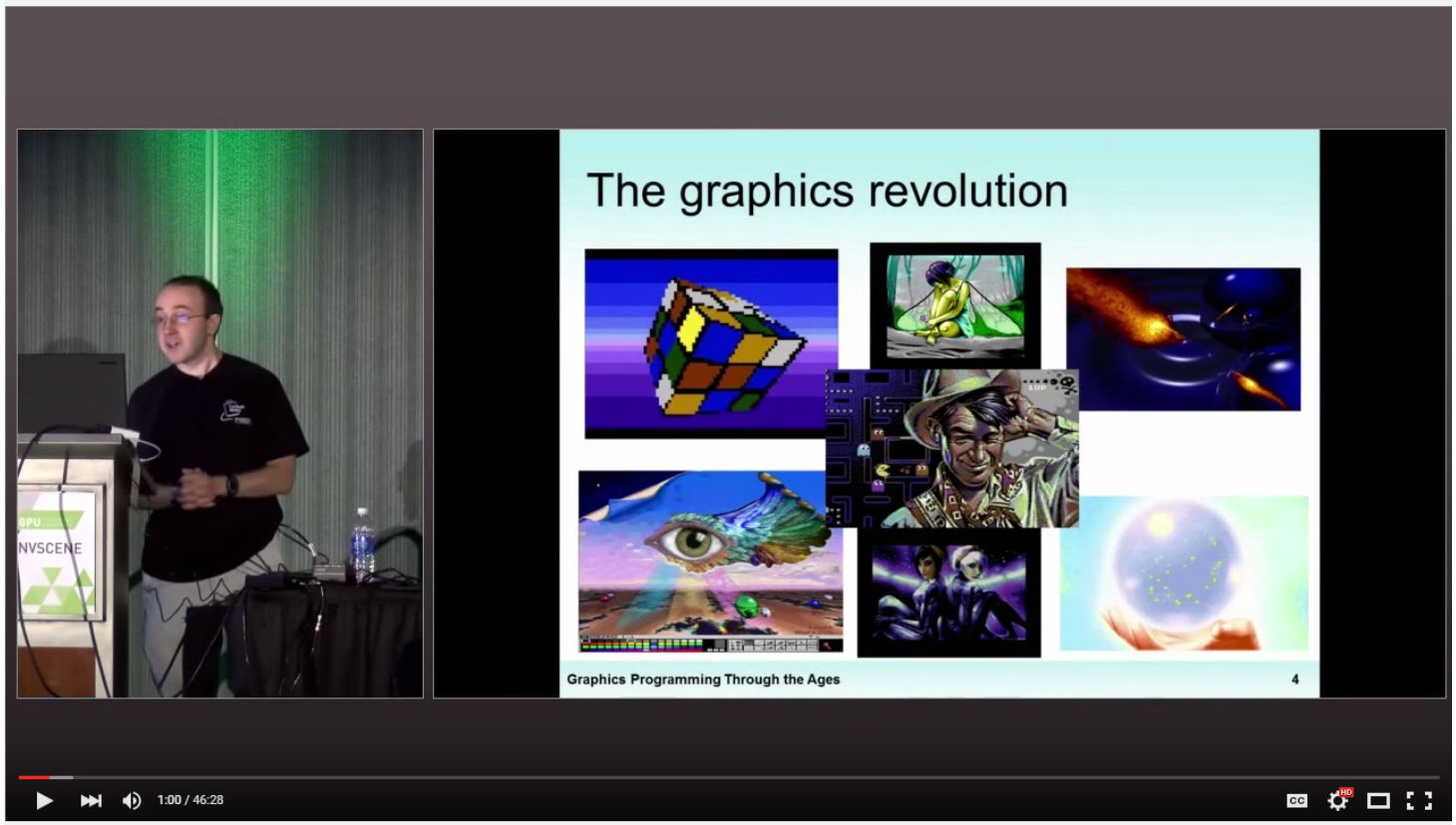
Turner Whitted: An Improved Illumination Model for Shaded Display.
Communications of the ACM 23, 6 (June 1980): 343–349

Rasterisation



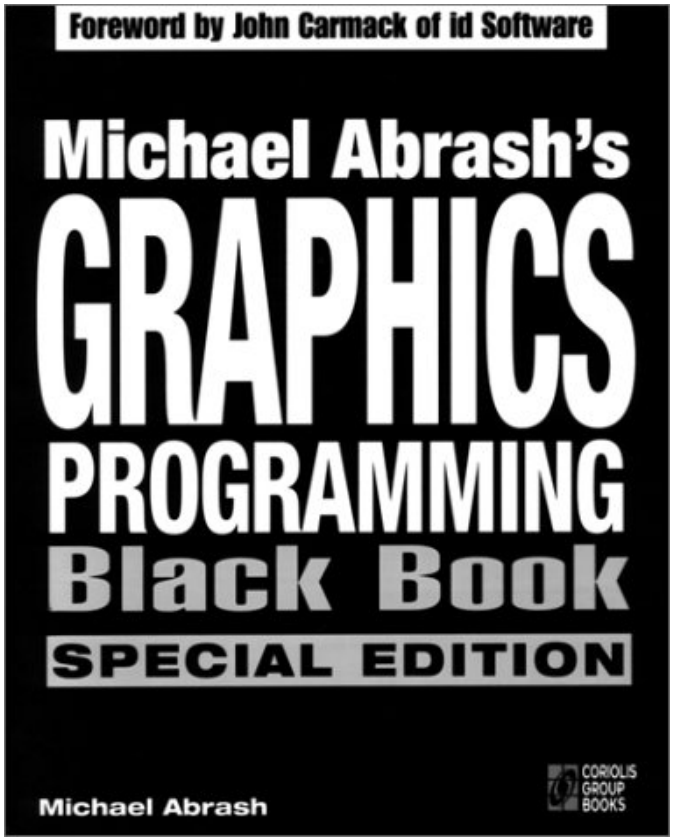
Zarch, 1987

Side note: If you find this interesting…



NVScene 2015 Session: Graphics Programming Through the Ages (Michael Dille, Keith Bare)
https://www.youtube.com/watch?v=D6DCEeJzZso

Side note #2



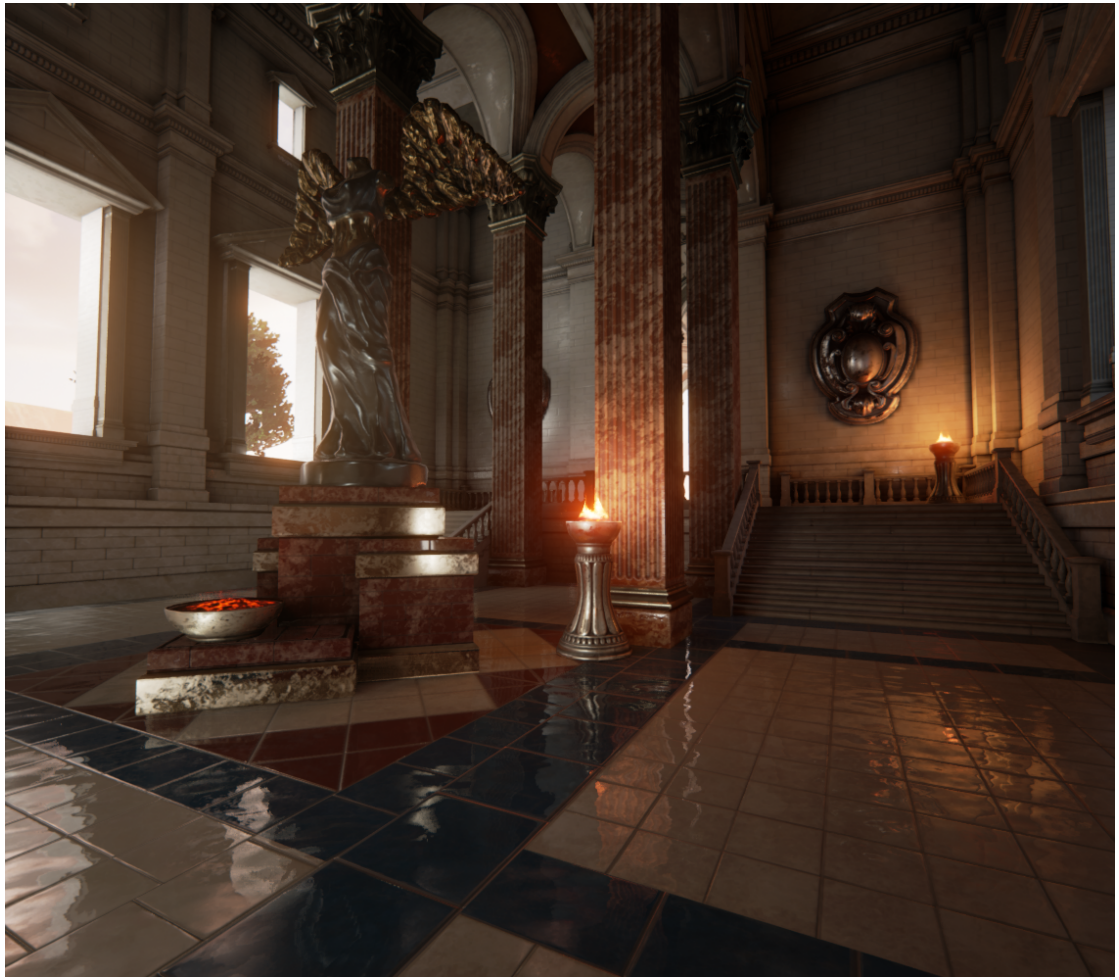
Michael Abrash's Graphics Programming Black Book
http://www.jagregory.com/abrash-black-book/

Rasterisation

Ultima Underworld: The Stygian Abyss, 1992

Rasterisation & Raytracing

TECHNISCHE UNIVERSITÄT DARMSTADT



Sun Temple – UE4 demo, 2014

Rasterisation

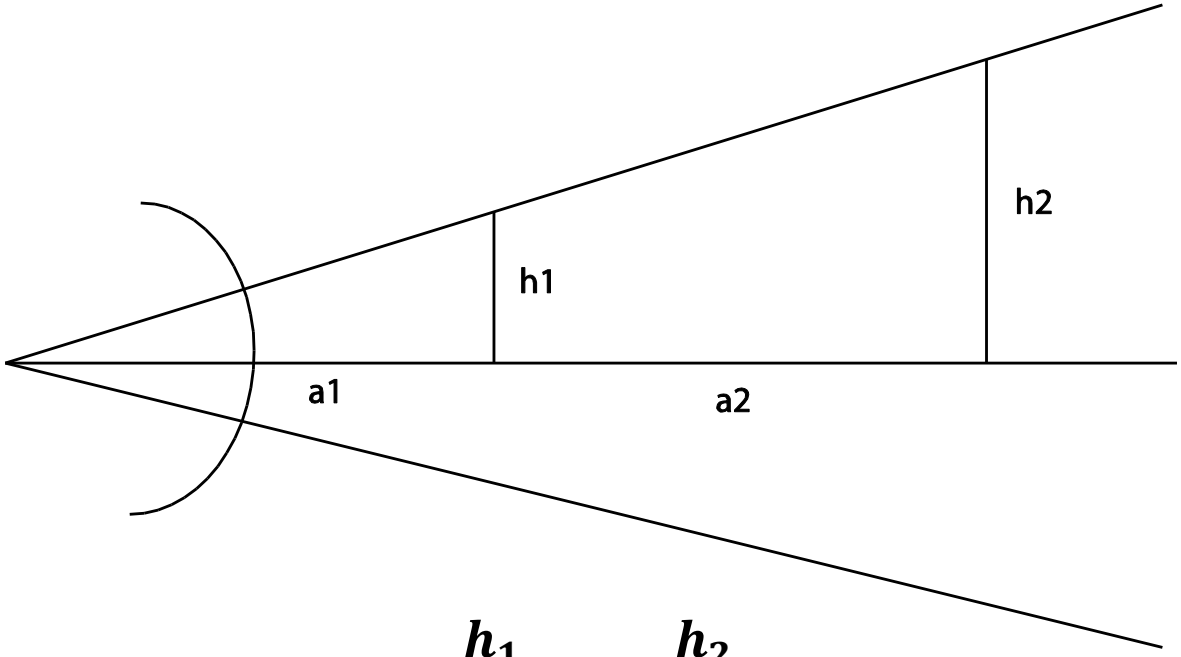
TECHNISCHE UNIVERSITÄT DARMSTADT

World defined by triangles

- Each triangle → three 3D points

```
foreach (tri in world) {
        Point p1 = transform(tri._1);
        Point p2 = transform(tri._2);
        Point p3 = transform(tri._3);
        drawTriangle(p1, p2, p3); // 2D operation
}
```

Perspective



$$\frac{h_1}{a_1} = \frac{h_2}{(a_1 + a_2)}$$

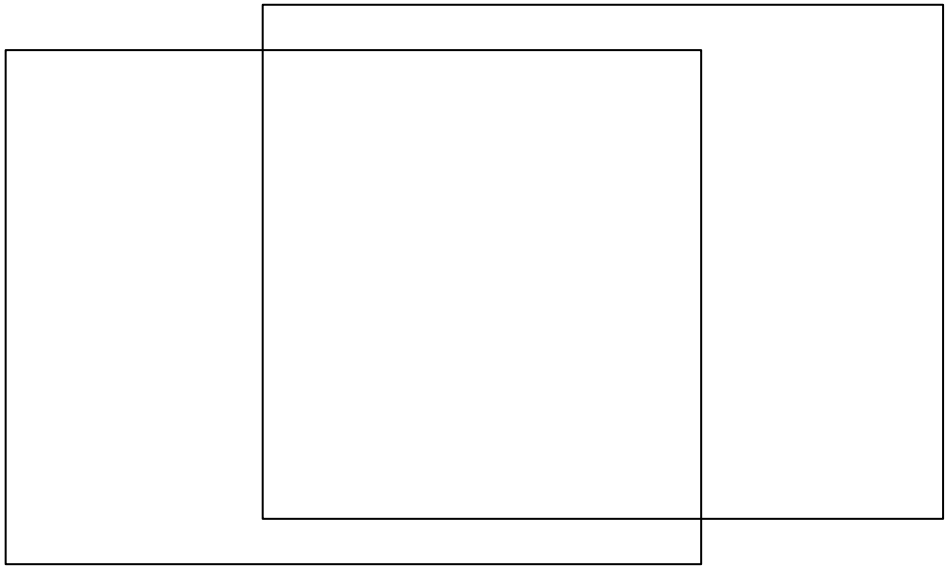
Doubled distance + doubled size → same projection

$$X_{proj} = \frac{z_{min}}{\text{distance}} X$$

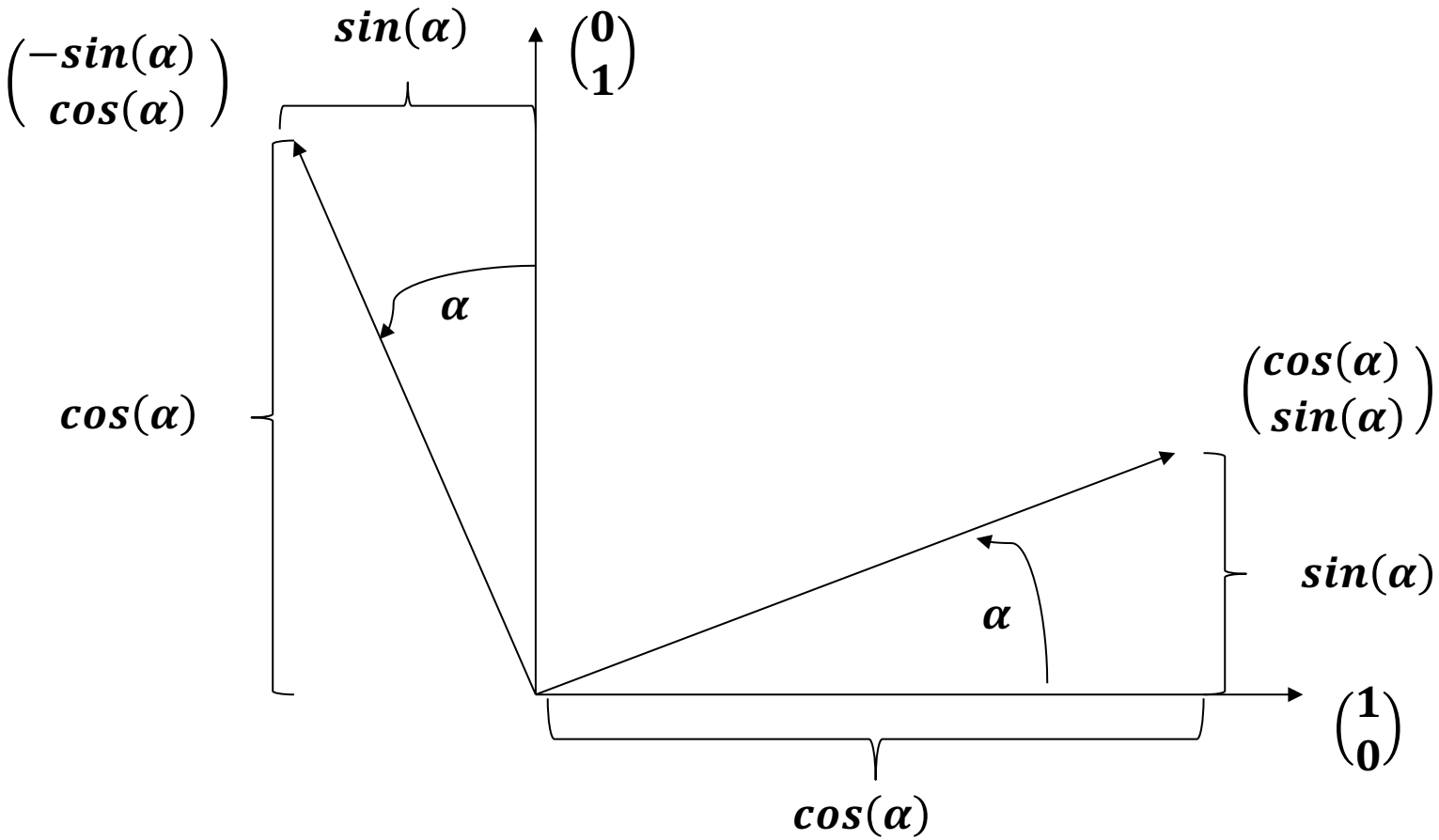
Offset from camera

$$X_{proj} = \frac{z_{min}}{\text{distance}}(X - x_{camera}) + \frac{screenWidth}{2}$$

$$Y_p = \ldots$$



Camera Rotations



Camera Rotations

TECHNISCHE UNIVERSITÄT DARMSTADT

Old Point

$$\begin{pmatrix} x \\ y \end{pmatrix} = x \begin{pmatrix} 1 \\ 0 \end{pmatrix} + y \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

New Point

$$R\left(\begin{pmatrix} x \\ y \end{pmatrix}, \alpha\right) = x \begin{pmatrix} cos(\alpha) \\ sin(\alpha) \end{pmatrix} + y \begin{pmatrix} -sin(\alpha) \\ cos(\alpha) \end{pmatrix}$$

$$R\left(\begin{pmatrix} x \\ y \end{pmatrix}, \alpha\right) = \begin{pmatrix} x \cdot cos(\alpha) \\ x \cdot sin(\alpha) \end{pmatrix} + \begin{pmatrix} -y \cdot sin(\alpha) \\ y \cdot cos(\alpha) \end{pmatrix}$$

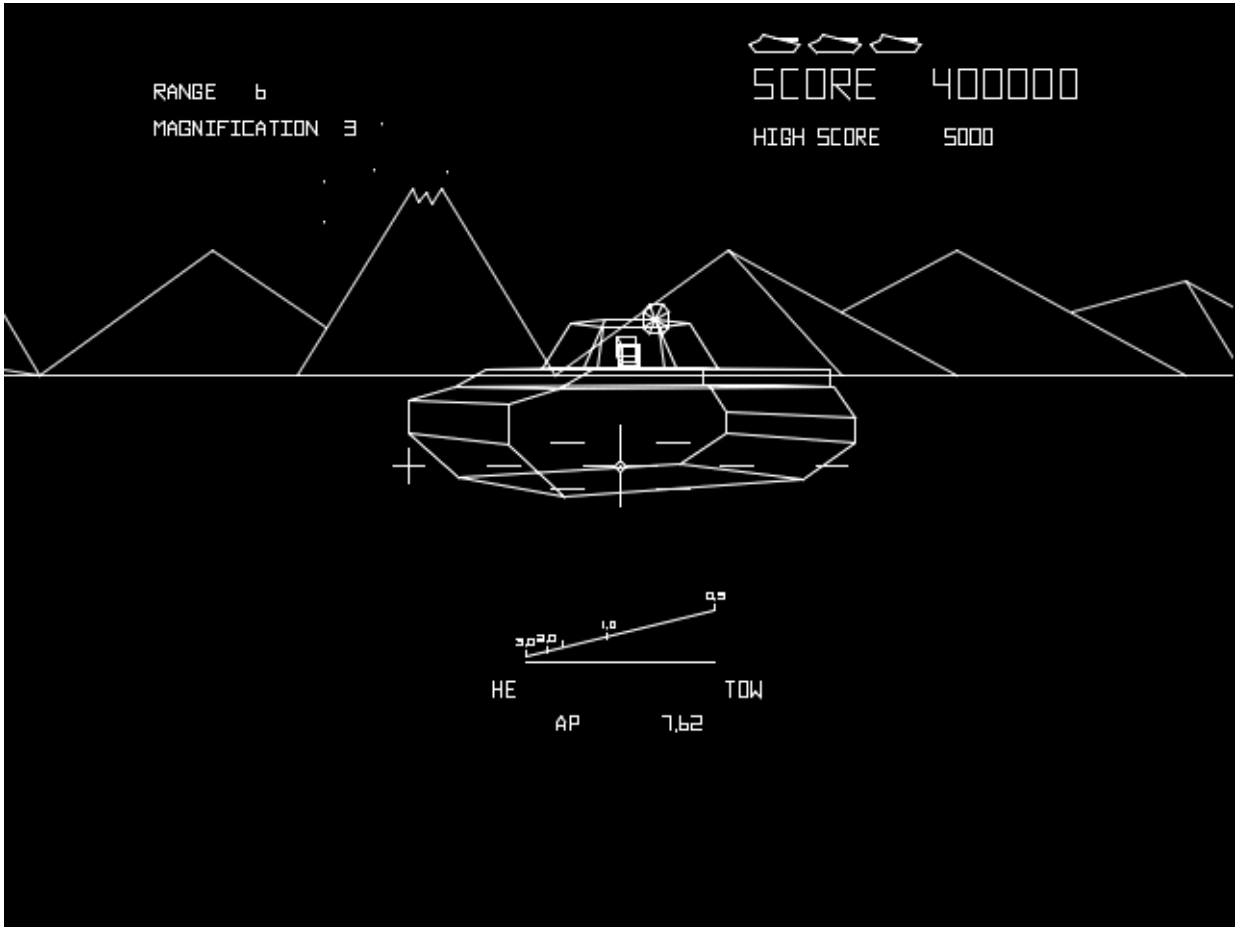
$$R\left(\begin{pmatrix} x \\ y \end{pmatrix}, \alpha\right) = \begin{pmatrix} x \cdot cos(\alpha) - y \cdot sin(\alpha) \\ x \cdot sin(\alpha) + y \cdot cos(\alpha) \end{pmatrix}$$

Camera Rotations

TECHNISCHE UNIVERSITÄT DARMSTADT

```
float dx = X - camera.x;
float dy = Y - camera.y;
float dz = Z - camera.z;
float d1x = cos(camera.ry) * dx + sin(camera.ry) * dz;
float d1y = dy;
float d1z = cos(camera.ry) * dz - sin(camera.ry) * dx;
float d2x = d1x;
float d2y = cos(camera.rx) * d1y - sin(camera.rx) * d1z;
float d2z = cos(camera.rx) * d1z + sin(camera.rx) * d1y;
float d3x = cos(camera.rz) * d2x + sin(camera.rz) * d2y;
float d3y = cos(camera.rz) * d2y – sin(camera.rz) * d2x;
float d3z = d2z;
Xp = (zmin / d3z) * d3x + screenWidth / 2;
Yp = …
```

Lines



Battlezone, 1980

Digital differential analyzer

TECHNISCHE UNIVERSITÄT DARMSTADT

```
dx = x2 - x1
dy = y2 - y1
for x from x1 to x2 {
    y = y1 + dy * (x - x1)
    plot(x, y)
}
```

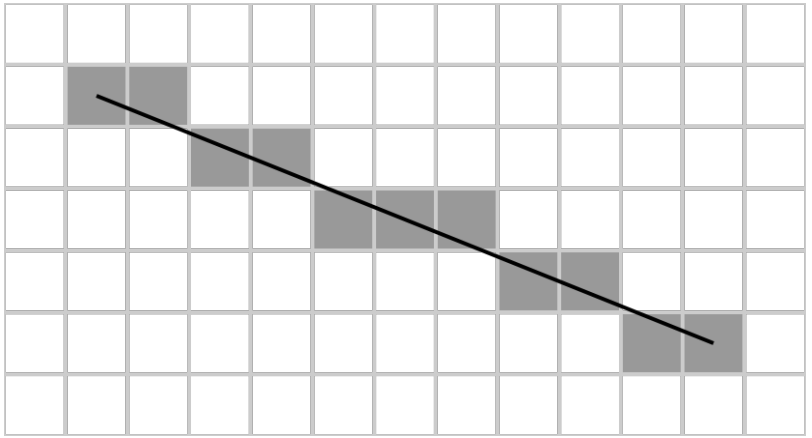
Bresenham

TECHNISCHE UNIVERSITÄT DARMSTADT

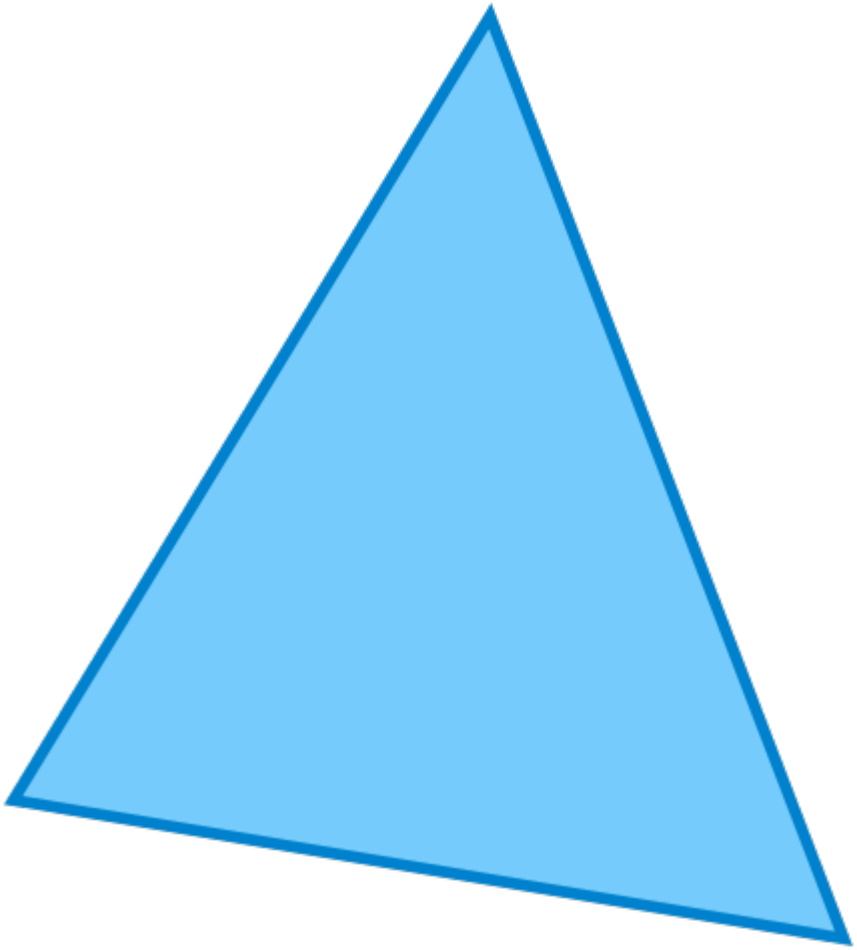
```
plotLine(x0,y0, x1,y1)
  dx=x1-x0
  dy=y1-y0

  D = 2*dy - dx
  plot(x0,y0)
  y=y0

  for x from x0+1 to x1
    D = D + (2*dy)
    if D > 0
      y = y+1
      D = D - (2*dx)
    plot(x,y)
```



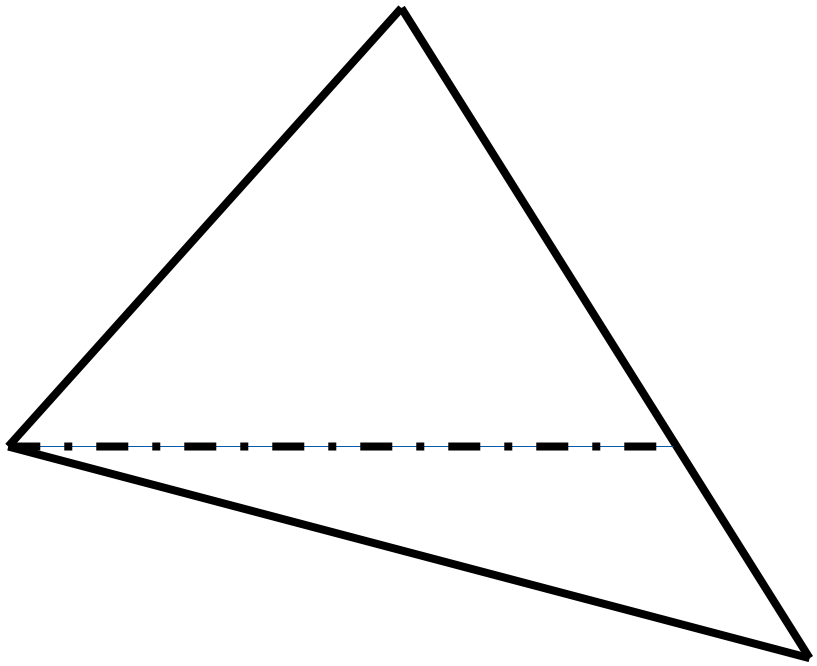
Triangles



Triangle Rasterisation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Find edge longest with biggest ydif
- Fill lines between long edge and other edge 1
- Fill lines between long edge and other edge 2



Mesh structure

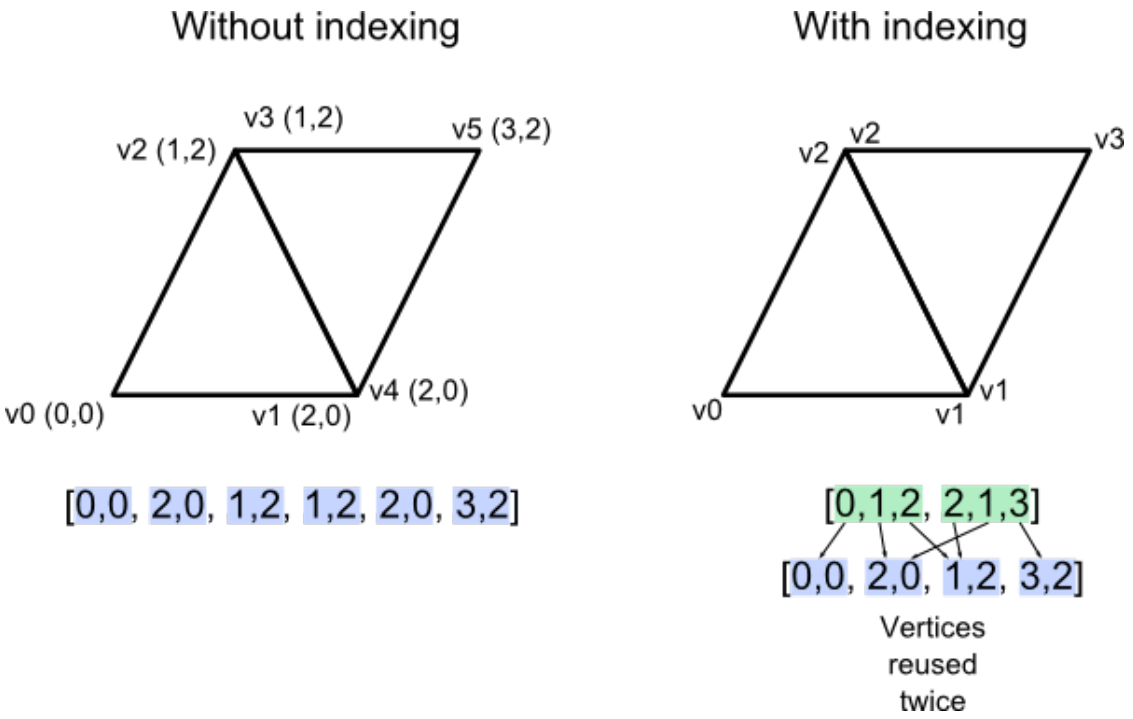
TECHNISCHE UNIVERSITÄT DARMSTADT

Array of 3D positions

- Often additional data

Array of indices

- Three indices -> triangle



Source: http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-9-vbo-indexing/

Example from Exercise‘s „SimpleGraphics“

TECHNISCHE UNIVERSITÄT DARMSTADT

```
VertexStructure structure;
structure.add("pos", Float3VertexData);
structure.add("tex", Float2VertexData);
vb = new VertexBuffer(4, structure, 0);
float* v = vb->lock();
{
  int i = 0;
  v[i++] = -1; v[i++] = 1; v[i++] = 0.5; v[i++] = 0; v[i++] = 0;
  v[i++] = 1; v[i++] = 1; v[i++] = 0.5; v[i++] = xAspect; v[i++] = 0;
  v[i++] = 1; v[i++] = -1; v[i++] = 0.5; v[i++] = xAspect; v[i++] = yAspect;
  v[i++] = -1; v[i++] = -1; v[i++] = 0.5; v[i++] = 0; v[i++] = yAspect;
}
vb->unlock();
ib = new IndexBuffer(6);
int* ii = ib->lock();
{
  int i = 0;
  ii[i++] = 0; ii[i++] = 1; ii[i++] = 3; ii[i++] = 1; ii[i++] = 2; ii[i++] = 3;
} ib->unlock();
```

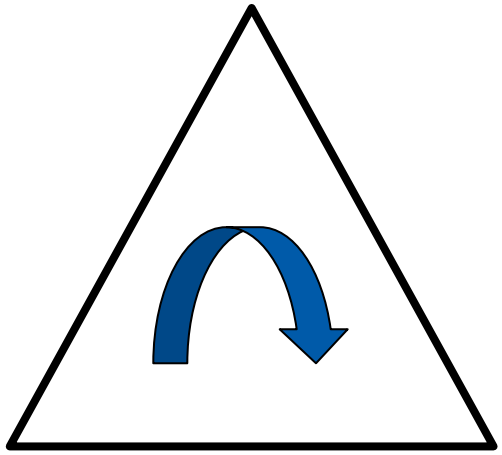
Backface Culling

TECHNISCHE UNIVERSITÄT DARMSTADT

Remove tris showing away from camera

Use tri winding

cross(b - a, c - a)



Dot product, Cross product

TECHNISCHE UNIVERSITÄT DARMSTADT

$$v_1 = (x_1, y_1, z_1), v_2 = (x_2, y_2, z_2)$$

Dot Product

$$v_1 \cdot v_2 = x_1x_2 + y_1y_2 + z_1z_2$$

$$v_1 \cdot v_2 = |v_1||v_2|\cos(\alpha)$$

Cross Product

$$v_1 \times v_2 = (y_1z_2 - z_1y_2, z_1x_2 - x_1z_2, x_1y_2 - y_1x_2)$$

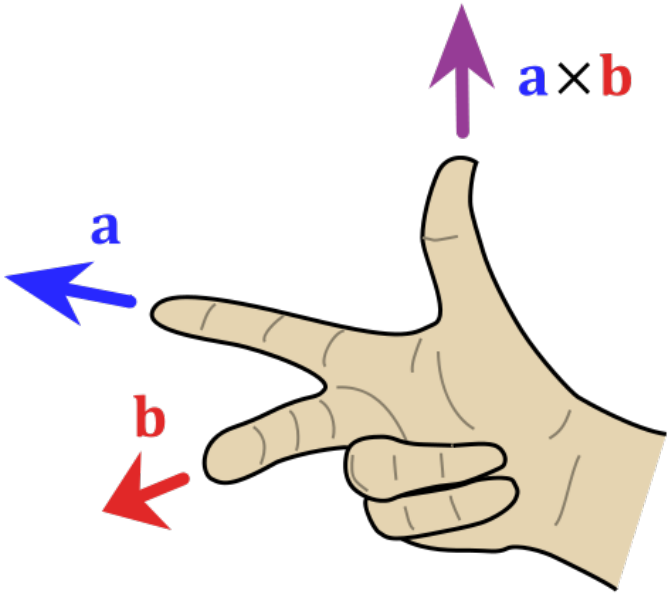
$$v_1 \times v_2 = |v_1||v_2|sin(\alpha)n$$

Geometric interpretations

TECHNISCHE UNIVERSITÄT DARMSTADT

Cross product

- Gives us the direction the normal is facing

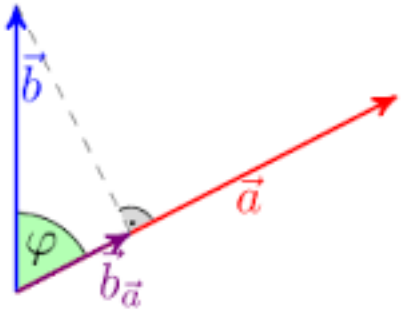


Dot product

- Projection of one vector onto the other

Special case of normals

- Can use to calculate angles



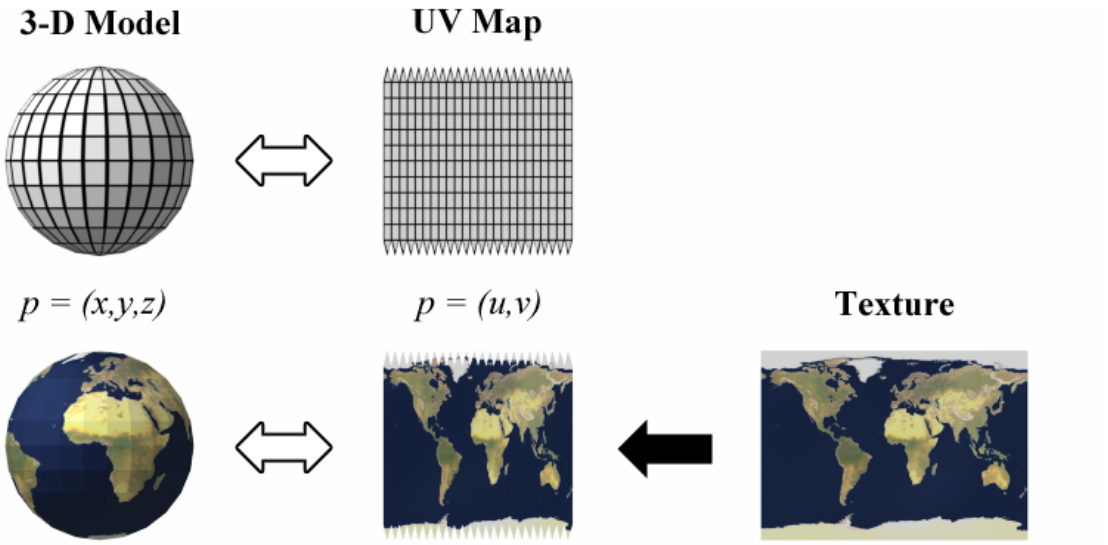
Textures

TECHNISCHE UNIVERSITÄT DARMSTADT

Add texture coordinates (uv) to mesh positions

Interpolate coordinates during rasterisation

Sample texture at interpolated coordinates



Depth Sorting

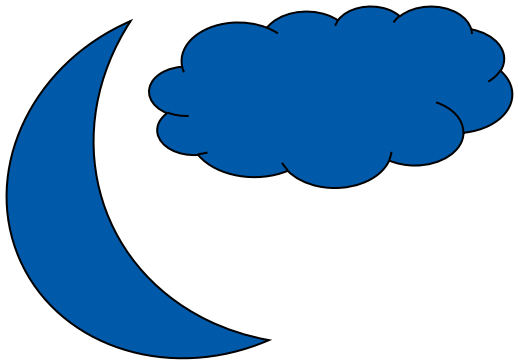
TECHNISCHE UNIVERSITÄT DARMSTADT

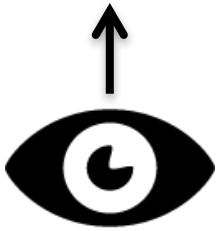
Sort only complete meshes

- For performance reasons

Sort by distance from camera to object

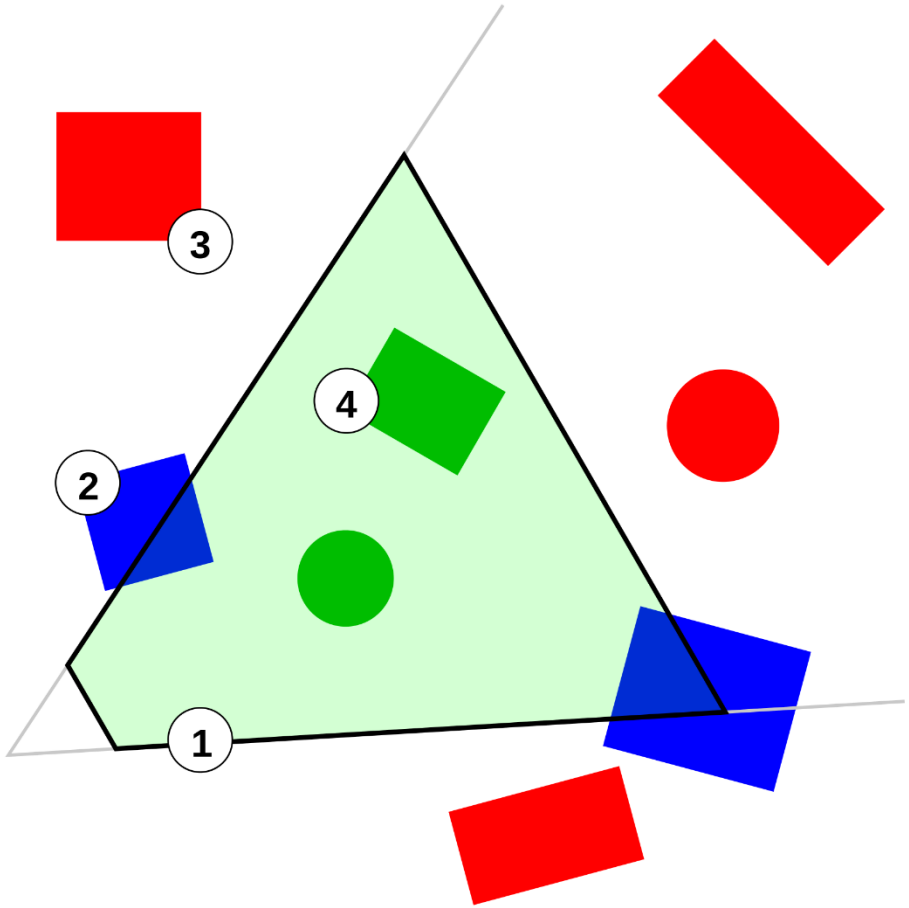
Draw nearest objects last





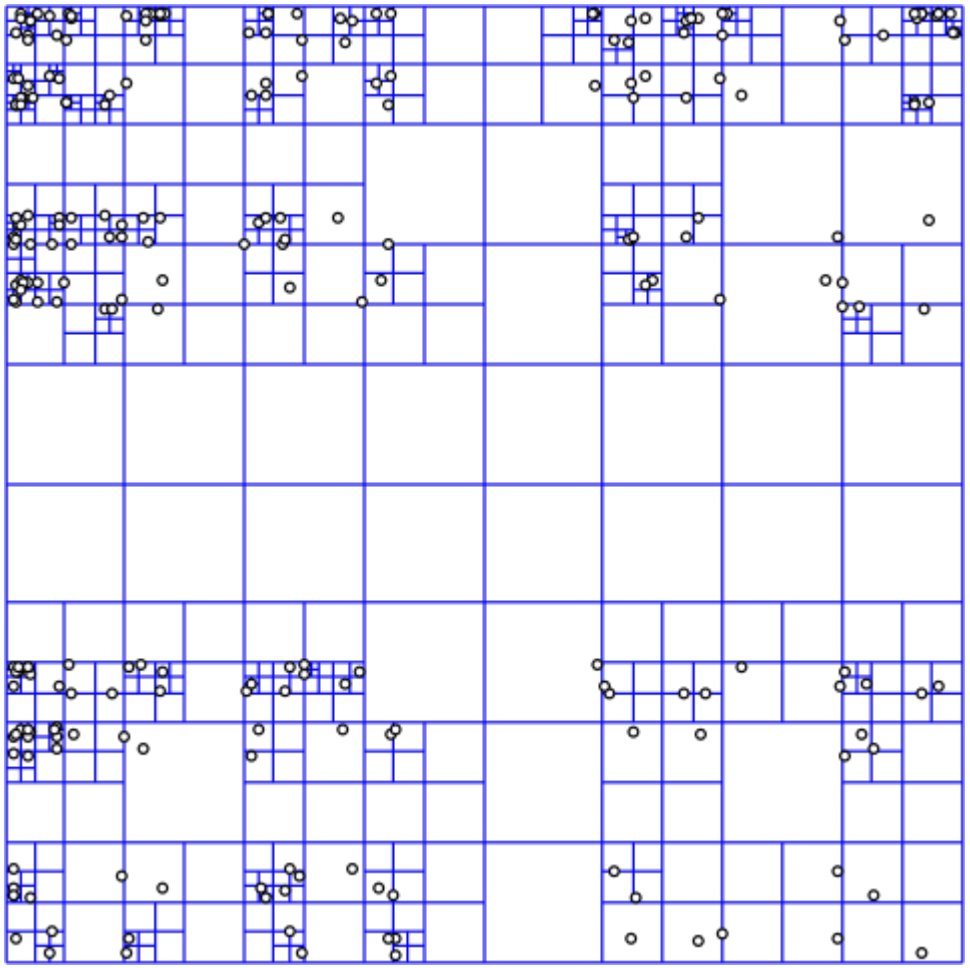
Frustum Culling

TECHNISCHE UNIVERSITÄT DARMSTADT

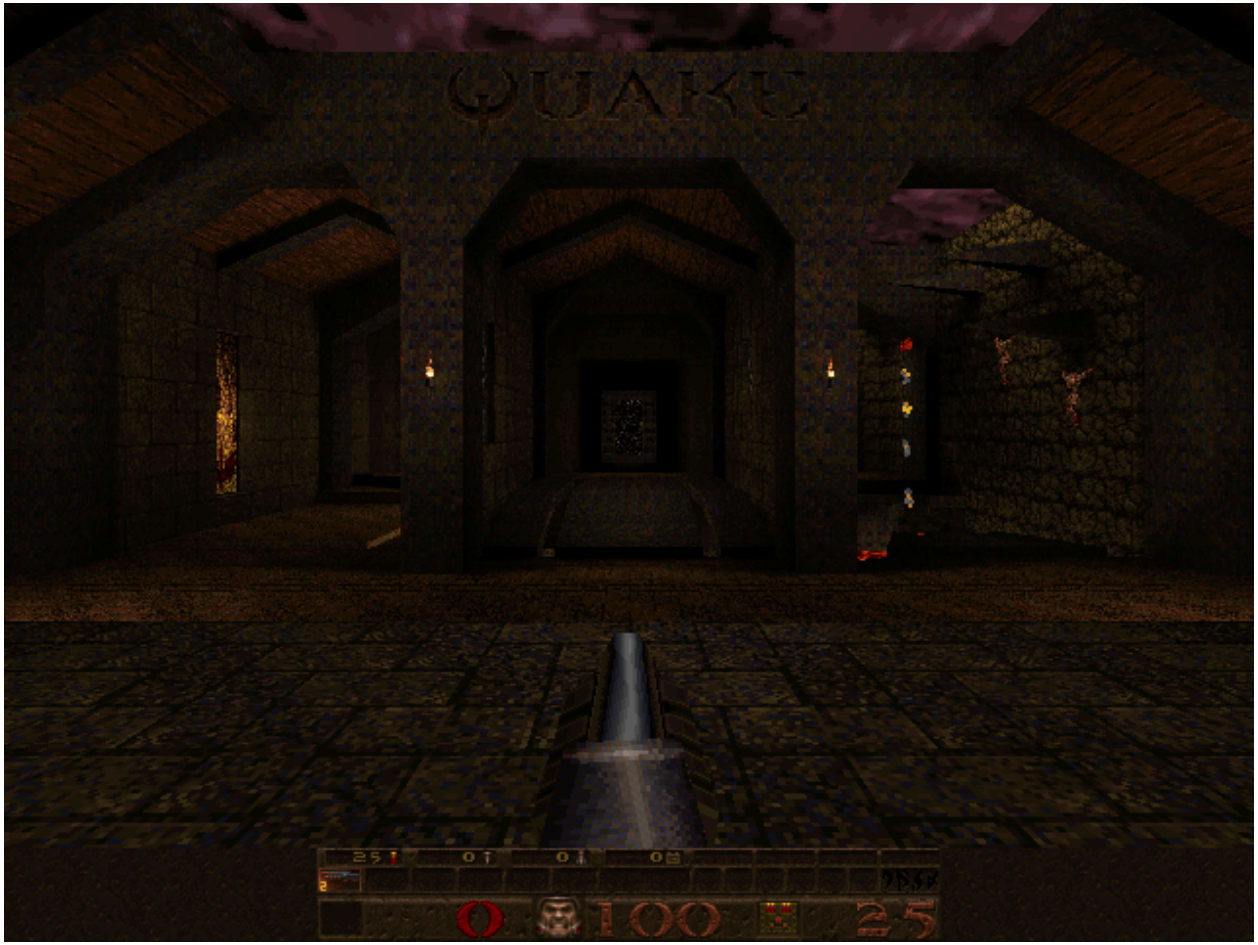


Hierarchical Scenes

TECHNISCHE UNIVERSITÄT DARMSTADT



Quake



Quake, 1996

Optimization

TECHNISCHE UNIVERSITÄT DARMSTADT

```
foreach (object in world) {
  if (infrustum(object)){
       render(object);
  }
}
```

Hierarchical structures slow on modern CPUs

- Cache misses

Occlusion Culling

TECHNISCHE UNIVERSITÄT DARMSTADT

Computations can be expensive

Precompute

- Unity

Rasterize in low resolution

- Software Occlusion Culling from intel

Occlusion Query

- Hardware feature

Remember: Games need steady performance

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

2D Rendering

- Positioning, Scaling, Rotating, Shearing

Raycasting vs. Raytracing

Triangle Rasterisation

Backface Culling

- Cross Product , Dot Product

Frustum Culling

Occlusion Culling

C++

Operator overloading

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class Number {
  …
};

Number operator+(Number a, Number b) {
  return …
}

Number a;
Number b;
Number c = a + b;
```

Operator overloading

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class Number {
 Number& operator++(); // ++num;
 Number operator++(int); // num++;
};

for (Number i = 0; i < 10; i++) { }

// maybe faster when ++ is overloaded
for (Number i = 0; i < 10; ++i) { }
```

Pre-increment: ++i

- Result is the value after incrementation

Post-increment: i++

- Result is the value before incrementation

References

TECHNISCHE UNIVERSITÄT DARMSTADT

New name for existing variable

Same syntax for access

```
int a = 3;
int& b = a;
b = 4;          // a == 4
```

References

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void swap(int& a, int& b) {
 int x = a;
 a = b;
 b = x;
}

int a = 3;
int b = 4;
Swap(a, b); // a == 4, b == 3
```

References

TECHNISCHE UNIVERSITÄT DARMSTADT

In theory just an unchangeable reference

- Not a hardware level concept
- Can often be removed by the optimizer

In practice works like restricted pointers

Added to support map implementations

```
class Map {
 int& operator[](int index);
};
```

Constness

TECHNISCHE UNIVERSITÄT DARMSTADT

```
const int a = 3;

a = 4 ;
```

Constness

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void bla(const int a) {
  a = 4;
}
```

Constness

TECHNISCHE UNIVERSITÄT DARMSTADT

```
const char* bla1 = "bla";

char* const bla2 = "bla";

bla1 = "blub";

bla2[0] = 'g';
```

Constness

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void bla(const int& a) {
  …
}
```

Hint for the compiler: Do what you want

- Copying a value can be faster than using a pointer

Constness

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class A {
 void method1() const {
      a = 3;
 }

 void method2() const {
      b = 3;
 }

 int a;
 mutable int b;
};
```

Constness

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class A {
 void method1() const;
 void method1();
};

A a;
a.method1();

const A b;
b.method1();
```

Templates

TECHNISCHE UNIVERSITÄT DARMSTADT

```
template<class T> class A {
 void method1() {

 }

 void method2();
};

A<int> a;
```

Templates

TECHNISCHE UNIVERSITÄT DARMSTADT

```
template<int i> void bla() { … }

bla<3>();
```

static

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// functions.h

void func1() { }

static void func2() { }

inline void func3() { }

namespace {
  void func4() { }
}
```

Ludum Dare 40

TECHNISCHE UNIVERSITÄT DARMSTADT

- 2-4th December at KOM
- Create a game in 72 hours
- Write us an eMail gamejam@kom.tu-darmstadt.de (max. 20 persons)

https://www.kom.tu-darmstadt.de/en/0/game-jams/?no_cache=1