Game Technology

Lecture 4 – 24.10.2015
Advanced Software Rendering

TECHNISCHE UNIVERSITÄT DARMSTADT

Dr.-Ing. Florian Mehm

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

Three Problems

Weird depth problems

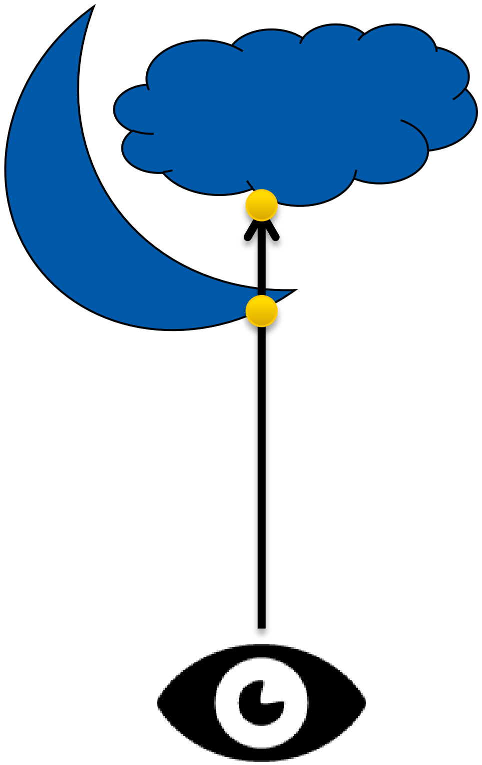
Weird textures

Weird rotations

Weird Depth Problems

Backface culling & object sorting can not handle

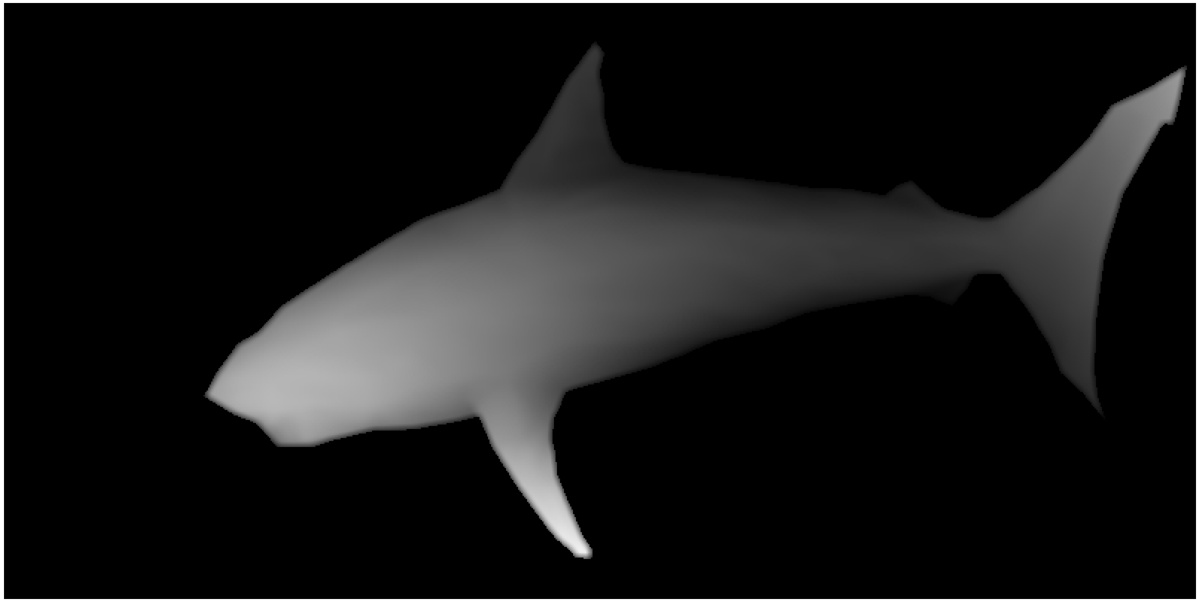
- Overlaping geometry
- Intersecting objects



Depth Buffer

TECHNISCHE UNIVERSITÄT DARMSTADT

```
foreach (pixel) {
        if (framebuffer[pixel.x, pixel.y].z < z) continue;
        framebuffer[pixel.x, pixel.y].rgb = rgb;
        framebuffer[pixel.x, pixel.y].z = z;
}
```



Depth Buffer

TECHNISCHE UNIVERSITÄT DARMSTADT

Dead Simple

Performance very bad…

- …when done in software

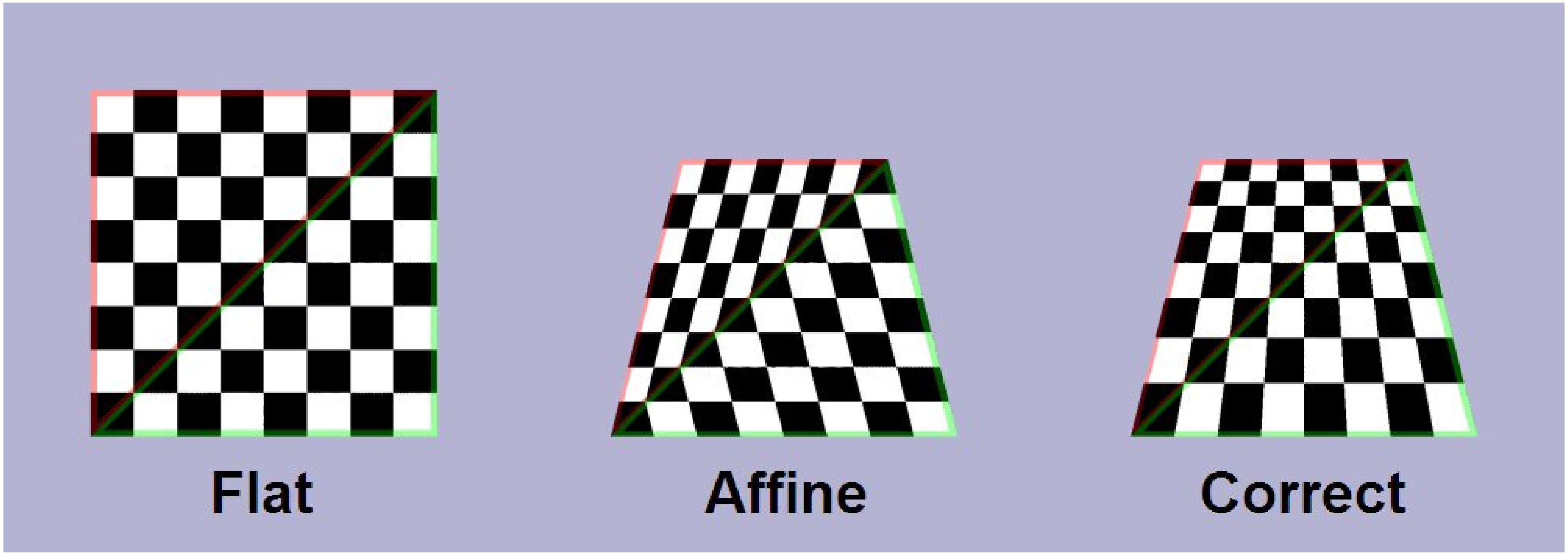
Performance OK…

- …when done in hardware

Does not help with partially transparent geometry

- Still only one z-value

Weird Textures



Weird Textures

TECHNISCHE UNIVERSITÄT DARMSTADT

WipEout, 1995

Perspective Texture Correction

TECHNISCHE UNIVERSITÄT DARMSTADT

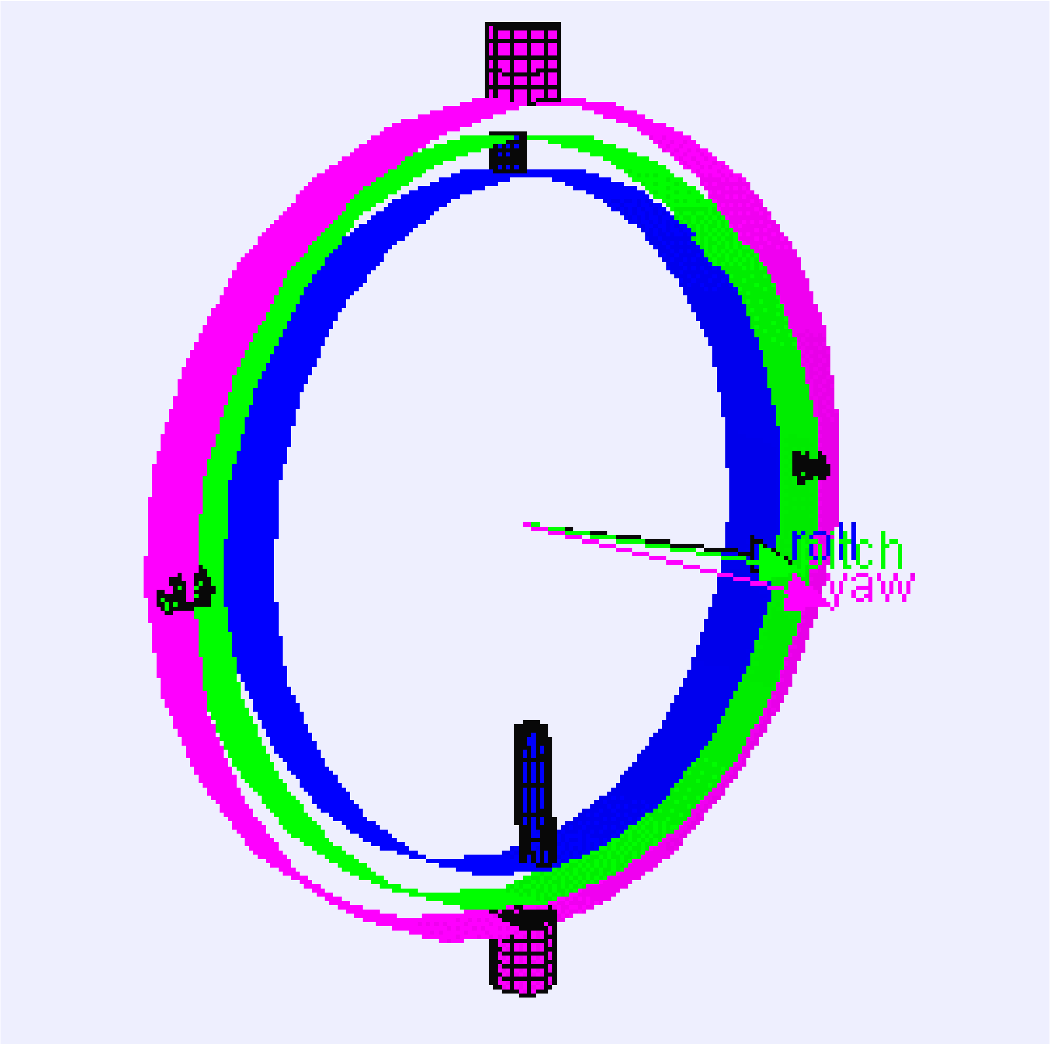
Regular interpolation

$$u = (1-\alpha)u_0 + \alpha u_1$$

Perspective correct interpolation

$$u_\alpha = \frac{(1-\alpha)\left(\frac{u_0}{z_0}\right) + \alpha\left(\frac{u_1}{z_1}\right)}{(1-\alpha)\left(\frac{1}{z_0}\right) + \alpha\left(\frac{1}{z_1}\right)}$$

Weird Rotations



Dependent on order

Rotate around x-axis
Rotate around y-axis
Rotate around z-axis

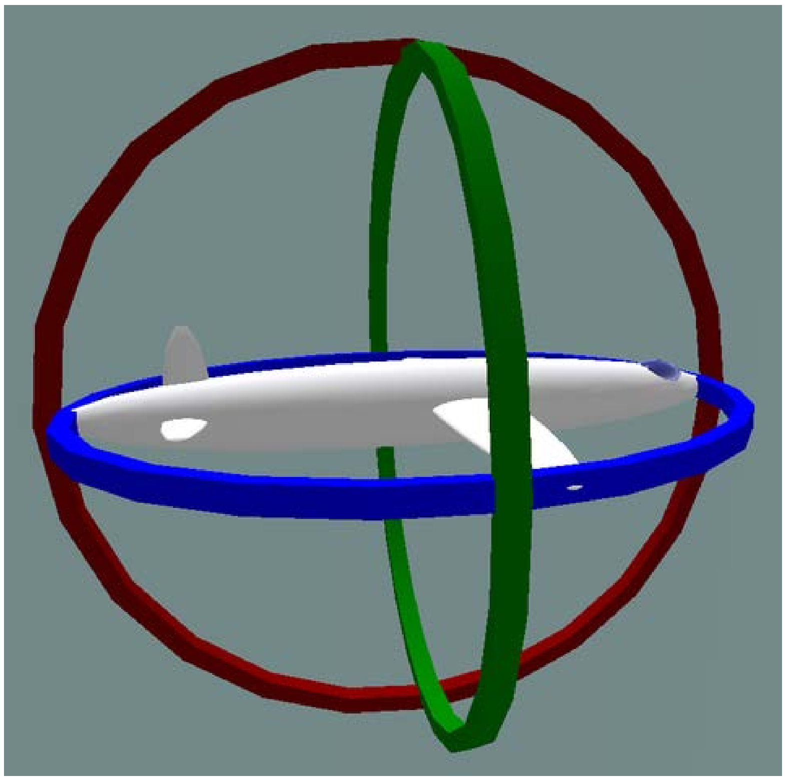
or

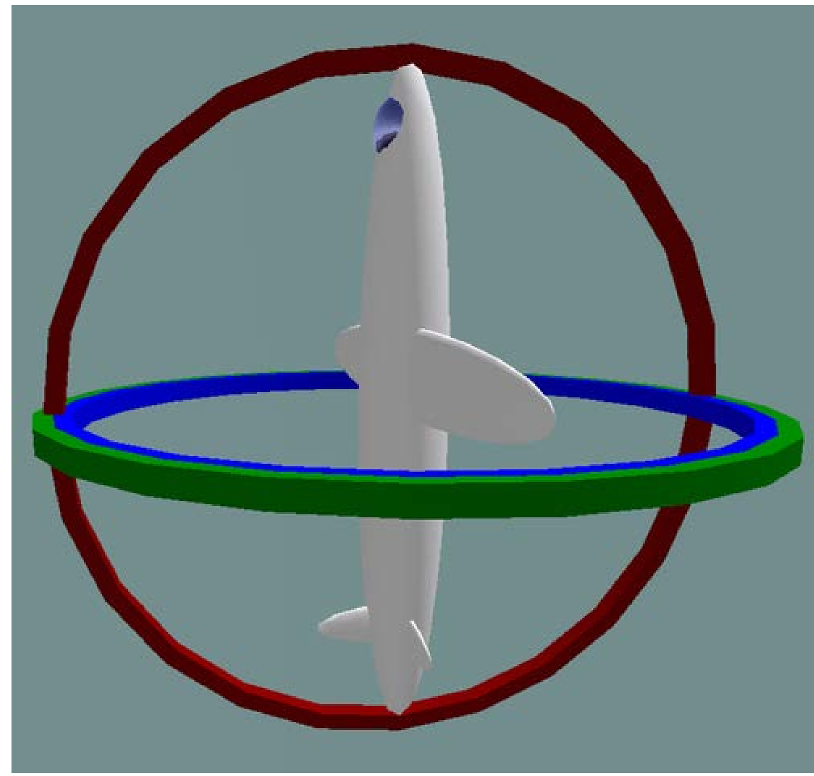
Rotate around z-axis
Rotate around y-axis
Rotate around x-axis

or

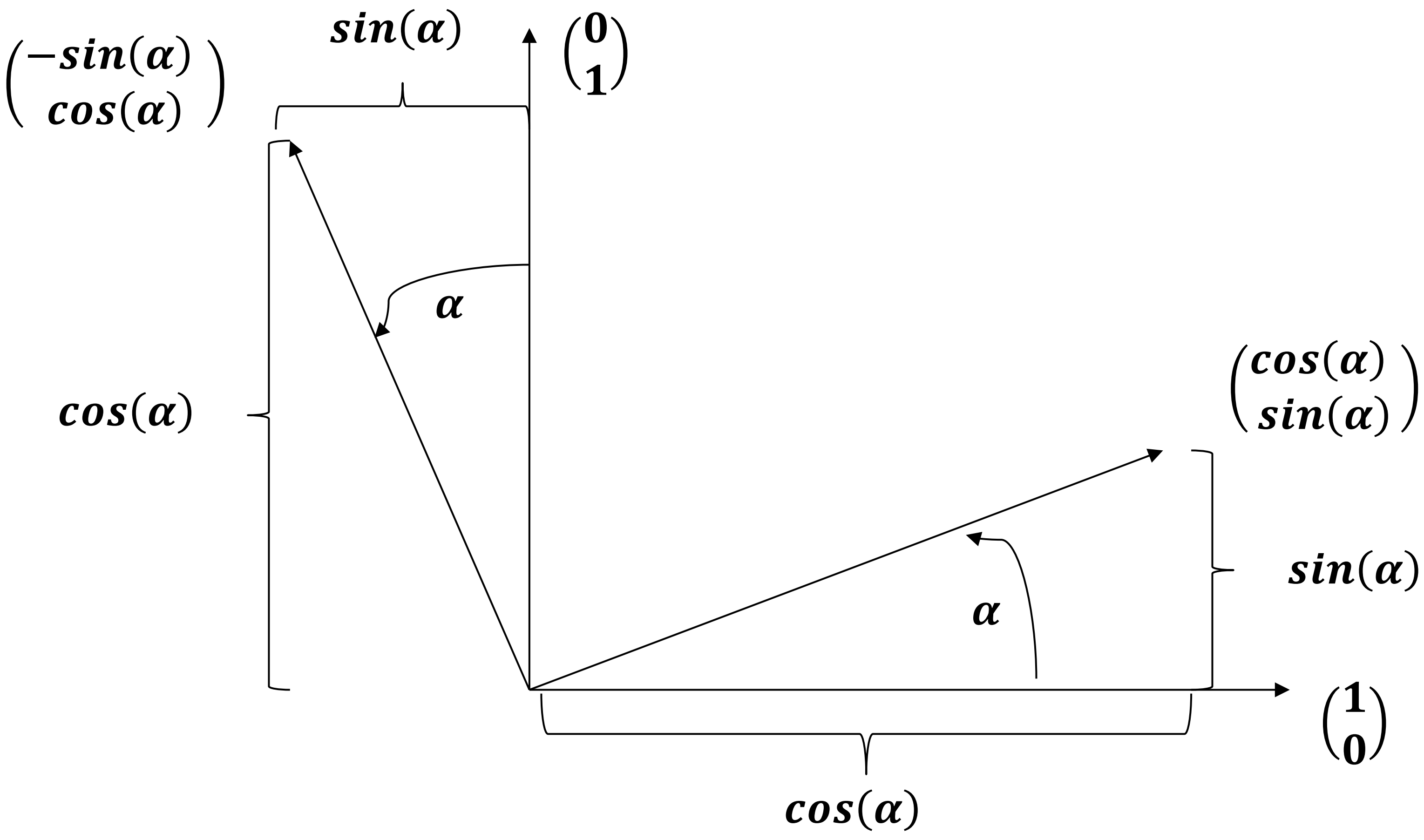
…

Gimbal Lock





Camera Rotations



Camera Rotations

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

Old Point

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

New Point

$$\begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

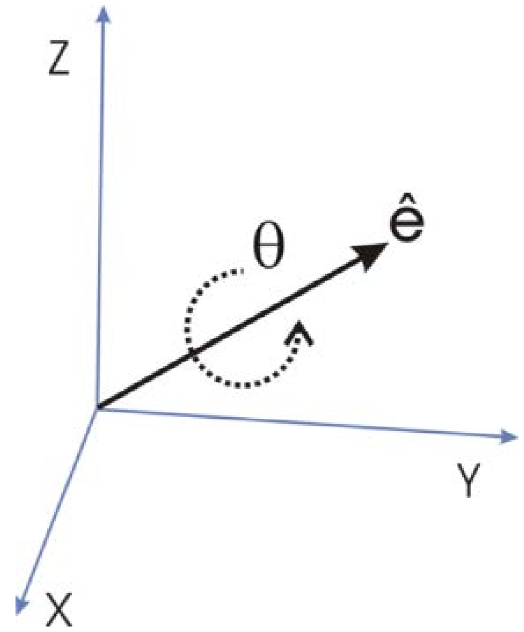
Matrix Multiplication

$$\begin{pmatrix} a & b & c \\ p & q & r \\ u & v & w \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ax + by + cz \\ px + qy + rz \\ ux + vy + wz \end{pmatrix}$$

4 Coordinates

Euler's rotation theorem:

Any rotation or sequence of rotations of a rigid body or coordinate system about a fixed point is equivalent to a single rotation by a given angle θ about a fixed axis (called Euler axis) that runs through the fixed point.



Rotation Matrix

u = unit vector

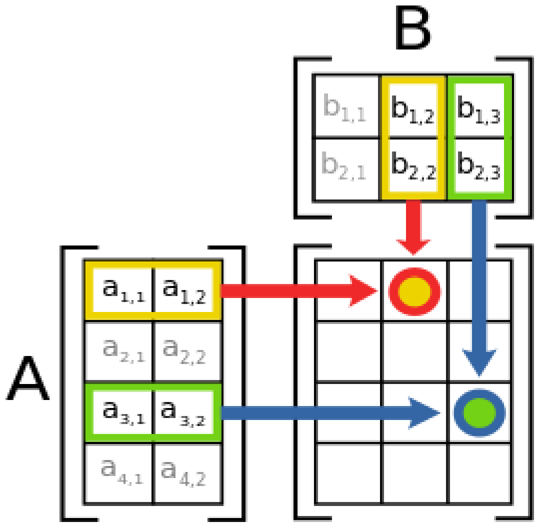
θ = rotation around $\boldsymbol{u}$

$$R = \begin{pmatrix} \cos(\theta) + u_x^2(1 - \cos(\theta) & u_x u_y(1 - \cos(\theta) - u_z \sin(\theta) & u_x u_z(1 - \cos(\theta)) + u_y \sin(\theta) \\ u_y u_x(1 - \cos(\theta) + u_z \sin(\theta) & \cos(\theta) + u_y^2(1 - \cos(\theta)) & u_y u_z(1 - \cos(\theta)) - u_x \sin(\theta) \\ u_z u_x(1 - \cos(\theta)) - u_y \sin(\theta) & u_z u_y(1 - \cos(\theta)) + u_x \sin(\theta) & \cos(\theta) + u_z^2(1 - \cos(\theta)) \end{pmatrix}$$

Matrix * Matrix

Concatenate Rotations

Save premultiplied matrices = Save calculations



$$x_{12} = a_{11}b_{12} + a_{12}b_{22}$$

$$x_{13} = a_{11}b_{13} + a_{12}b_{23}$$

$$x_{32} = a_{31}b_{12} + a_{32}b_{22}$$

$$x_{33} = a_{31}b_{13} + a_{32}b_{23}$$

Identity Matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Affine Transformations

Preserving

- Points
- Straing lines
- Planes

Translation

Scaling

Rotation

Shearing

Matrix Transformations

Dimension (2/3) * Dimension matrices support all affine transformations…

… except for translations

Translation Matrix

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Homogenous Coordinates

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \rightarrow \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{pmatrix}$$

3D Point: $\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ **3D Direction:** $\begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$

Would you like to know more?
GDC 2015 Talk by Squirrel Eiserloh
Slides available at: http://www.essentialmath.com/tutorial.htm

Perspective Projection

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

4x4 Matrix

$$\begin{pmatrix} \begin{pmatrix} v_x \end{pmatrix} & \begin{pmatrix} v_y \end{pmatrix} & \begin{pmatrix} v_z \end{pmatrix} & \begin{pmatrix} v_t \end{pmatrix} \\ (& v_p &) & \mathbf{1} \end{pmatrix}$$

Typical Setup

projection * view * model * position

Watch out for rotation order (column-major vs. row-major)

Projection

- Kore::Matrix::perspectiveProjection
  - Field of view
    - Aspect ratio (width / height)
    - z near, z far

View

- Kore::Matrix::lookAt
  - Eye vector
  - At vector
  - Up vector

Model

- Translations, Rotations,…

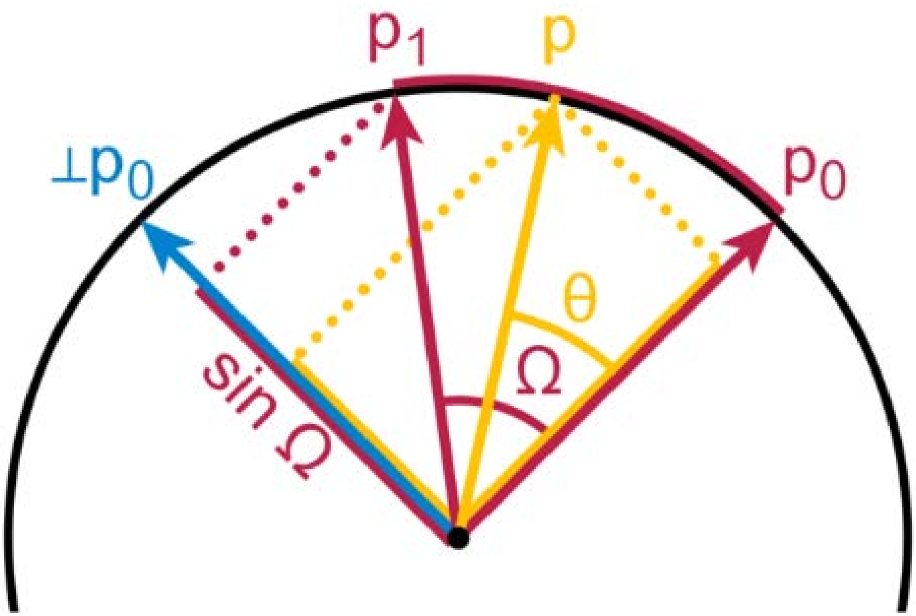
Rotation interpolation

Euler angles

- Easy for one rotation
- Super weird for three rotations

Rotation matrices

- Difficult
- Instable



Quaternions

TECHNISCHE UNIVERSITÄT DARMSTADT

4D imaginary numbers

Three imaginary components

$$i^2 = j^2 = k^2 = ijk = -1$$

Can represent rotations

$$q = e^{\frac{\theta}{2}(u_x i + u_y j + u_z k)} = cos\left(\frac{\theta}{2}\right) + (u_x i + u_y j + u_z k_x) sin(\frac{\theta}{2})$$

Rotation

$$v_{rot} = qvq^{-1}$$

Quaternions

$$(\boldsymbol{w}, \boldsymbol{v})$$

w: real scalar

v: Imaginary vector (x, y, z)

$$\boldsymbol{q_1 q_2 = (w_1 w_2 - v_1 \cdot v_2, \mathrm{v}_1 \times v_2 + w_1 v_2 + w_2 v_1)}$$

$$\boldsymbol{q_1 q_2 \neq q_2 q_1}$$

Inverse

$$\boldsymbol{q_1^{-1} = \frac{w, -v}{(w^2 + v_x^2 + v_y^2 + v_z^2)}}$$

Interpolation

Spherical Linear intERPolation (SLERP)

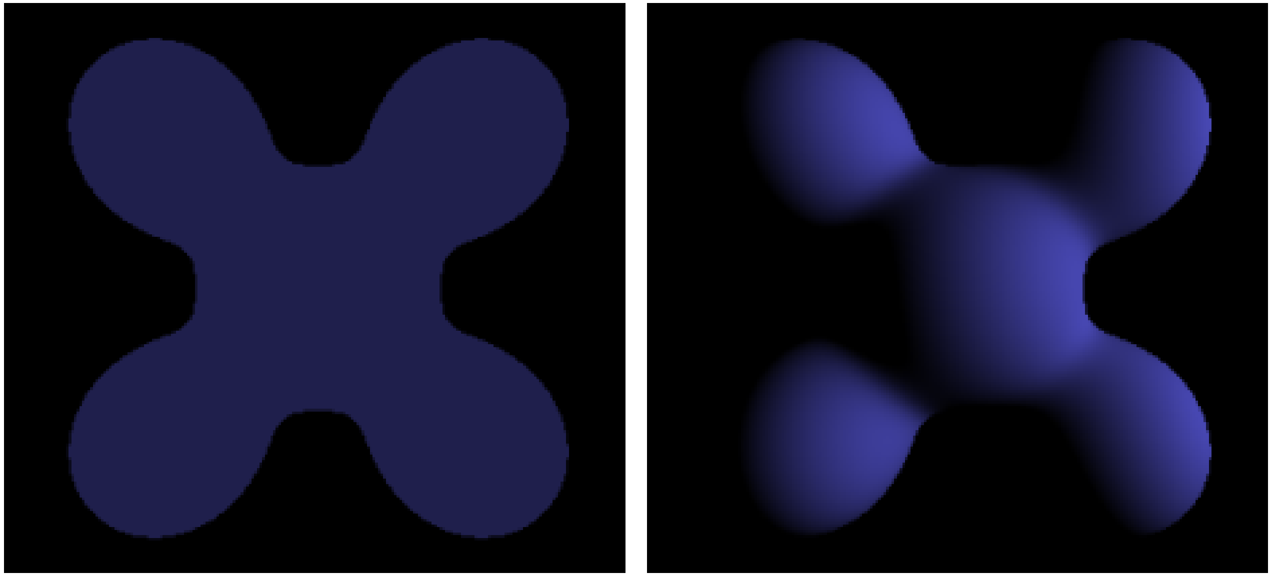
$$slerp(q_1, q_2, t) = \frac{sin((1-t)\theta)}{sin(\theta)} q_1 + \frac{sin(t\theta)}{sin(\theta)} q_2$$

$$\theta = \frac{cos^{-1}\left(w_1 w_2 + v_{x,1} v_{x,2} + v_{y,1} v_{y,2} + v_{z,1}\, v_{z,2}\right)}{|q_1||q_2|}$$

Quaternion to Matrix

$$\begin{pmatrix} 1-2y^2-2z^2 & 2xy-2zw & 2xz+2yw \\ 2xy+2zw & 1-2x^2-2z^2 & 2yz-2xw \\ 2xz-2yw & 2yz+2xw & 1-2x^2-2y^2 \end{pmatrix}$$

Lighting



Normals

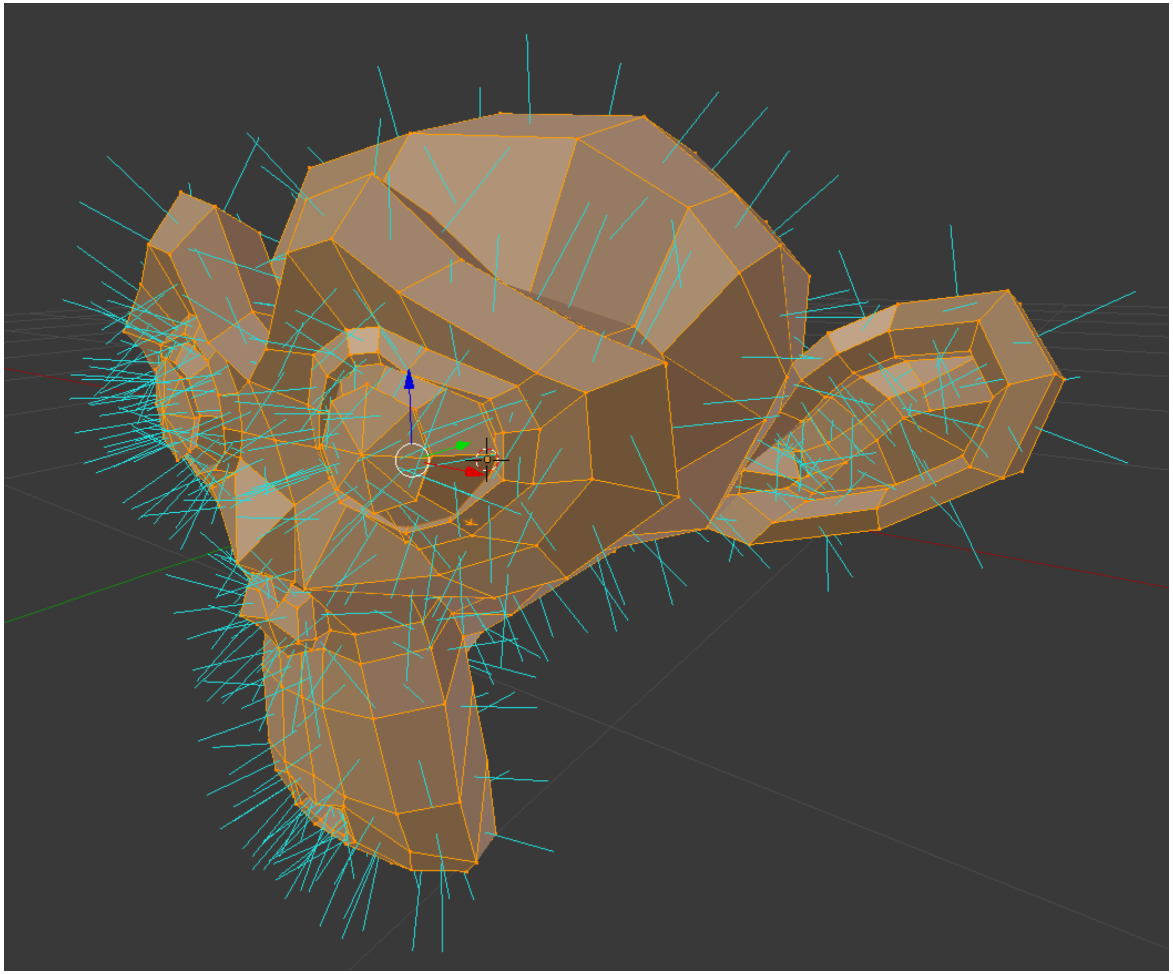
Defined per vertex

Direction: $\mathbf{n} = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$

Translation * n = n

Rotation * n = (…)

Normals



Vertex Splits

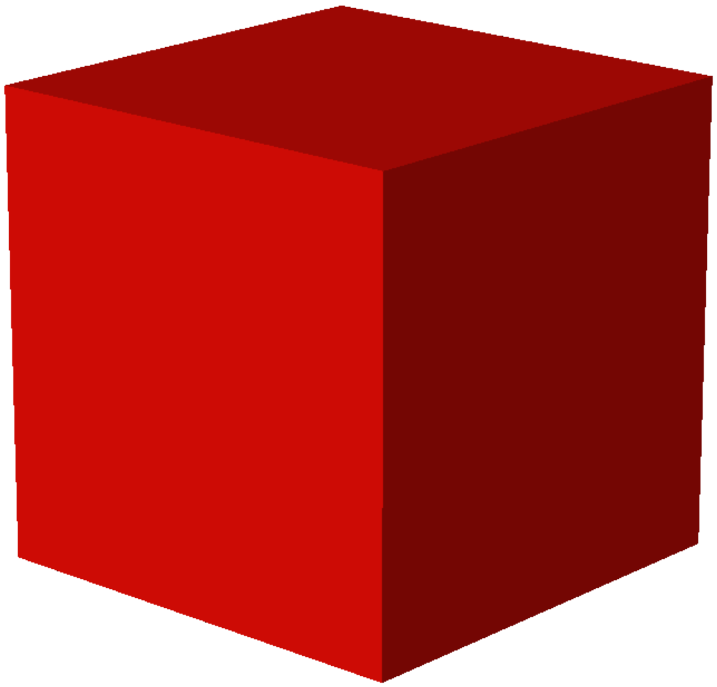
We will be saving normals per vertex

- During calculation, smoothe between the normals

What if we want sharp corners?

Every vertex needs to have several normals

- Either split the mesh during exporting
- Or during importing, create several vertices for the same position for each different normal

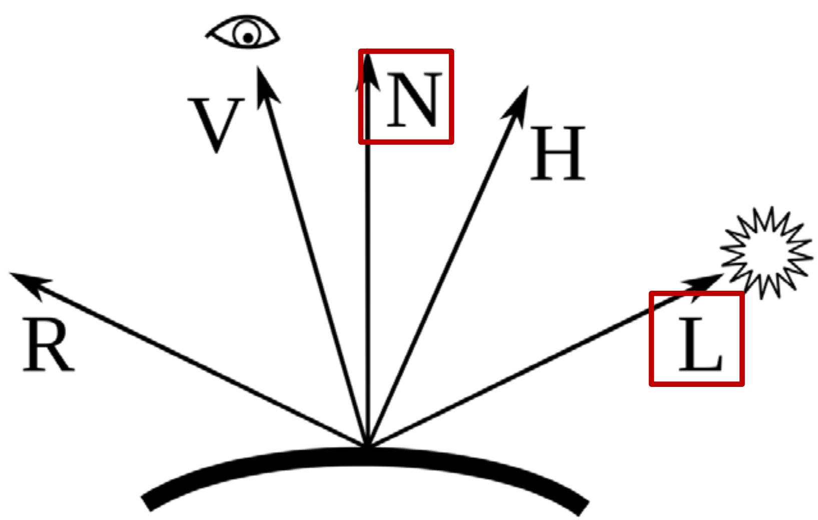


Super Basic Lighting

L = Light Direction (normalized vector towards light)

N = Normal

$$intensity = L \cdot N$$



Super Basic Lighting

$$\boldsymbol{intensity = L \cdot N}$$

$$\boldsymbol{v_1 \cdot v_2 = |v_1| \cos(\alpha)}$$

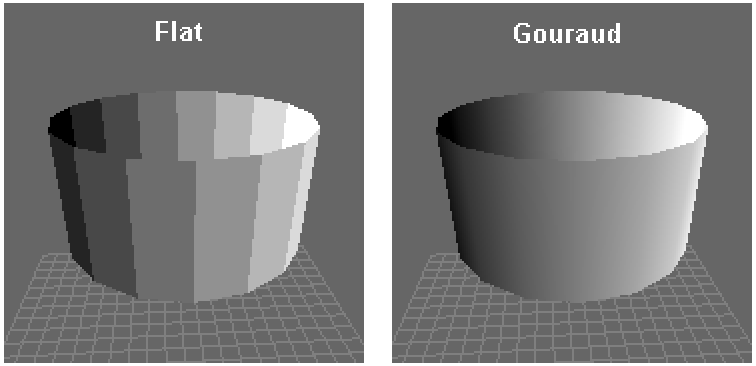
$$\boldsymbol{intensity = \cos(\alpha)}$$

Intensity is depending on the angle between L and N

Per Vertex vs per Pixel

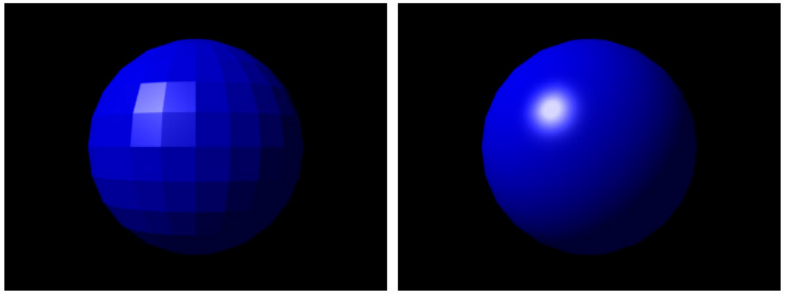
Per Vertex

- Fast
- Calculate lighting per vertex
- Interpolate colors
- → Gouraud shading



Per Pixel

- Pretty
- Interpolate normals
- Calculate lighting per pixel
- →Phong shading



FLAT SHADING PHONG SHADING

Parallel Computations

Superscalar CPUs

SIMD Instructions

Multithreading

C/C++ Fail

No standardized support for SIMD instructions

Multithreading support since 2011

Superscalar Execution

```
c = a + b
d = a + b // can be parallelized

c = a + b
d = a + c // can not be parallelized
```

Superscalar Execution

No explicit support necessary (or even possible?)

Compiler can reorder instructions

Keep in mind when optimizing

- Profiler can show < 1 ticks per instruction

SIMD Instructions

SIMD – Single Instruction Multiple Data

- Apply same calculation to multiple values

Can easily be applied to Vector/Matrix math

Automatic compiler optimizations – very limited

x86

SSE – since Pentium 3 in 1999 (Streaming SIMD Extensions)

128 bit registers

- 4 float numbers per register

SSE2, SSE3, SSE4, AVX,…

SSE2 supported by every x64 CPU

64 bit Operating Systems use SSE instructions for all floating point calculations

ARM

NEON

Since Cortex-A8 (but only optional)

128 bit registers

…

Intrinsics

```
#include <xmmintrin.h>
__m128 value1 = _mm_set_ps(1, 2, 3, 4);
__m128 value2 = _mm_set_ps(5, 6, 7, 8);
__m128 added = _mm_add_ps(value1, value2);
float allAdded = added.m128_f32[0] + added.m128_f32[1]
  + added.m128_f32[2] + added.m128_f32[3];
```

Just like assembler programming

- (minus register numbers)

Current Situation

No Standard

SSE and Neon – incompatible intrinsics

Different compilers – ~compatible instrinsics

Libraries of small functions of macros can help

- http://www.gamedev.net/page/resources/_/technical/general-programming/practical-cross-platform-simd-math-r3068

Multithreading

Standard support since 2011

OS APIs since 1980s

Kore::Thread

Multithreading

Traditionally avoided in Games

Very important for multicore CPUs

Multithreading

Independent execution threads

Same address space

Lots of problems

Use for speed

- Number of threads = number of cores

Use for asynchronicity

- E.g. Loading data from disk

Never use for convenience

Race Conditions

Race Conditions

Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

Race Conditions

Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

Race Conditions

Very difficult to debug

Might happen very rarely

Worst kind of bugs

Mutex

```
Kore::Mutex m;
m.Create();
m.Lock();
...
// access shared state
m.Unlock();
```

Mapped to mutex in Linux

Mapped to critical section in Windows

- Windows Mutex is used for interprocess sync

Mutex

TECHNISCHE UNIVERSITÄT DARMSTADT

Can slow down program

- Syscalls, cache flushes,…

Minimize sync points

Typical design a

- CPU core 1 only for physics
- CPU core 2 for everything else
  - Sync once per frame

Typical design b

- Work package objects
- Worker threads (one for each CPU core)
- Work package manager assigns packages to threads

Lock Free Multithreading

Can speed up programs

Atomic operations

Arcane magic