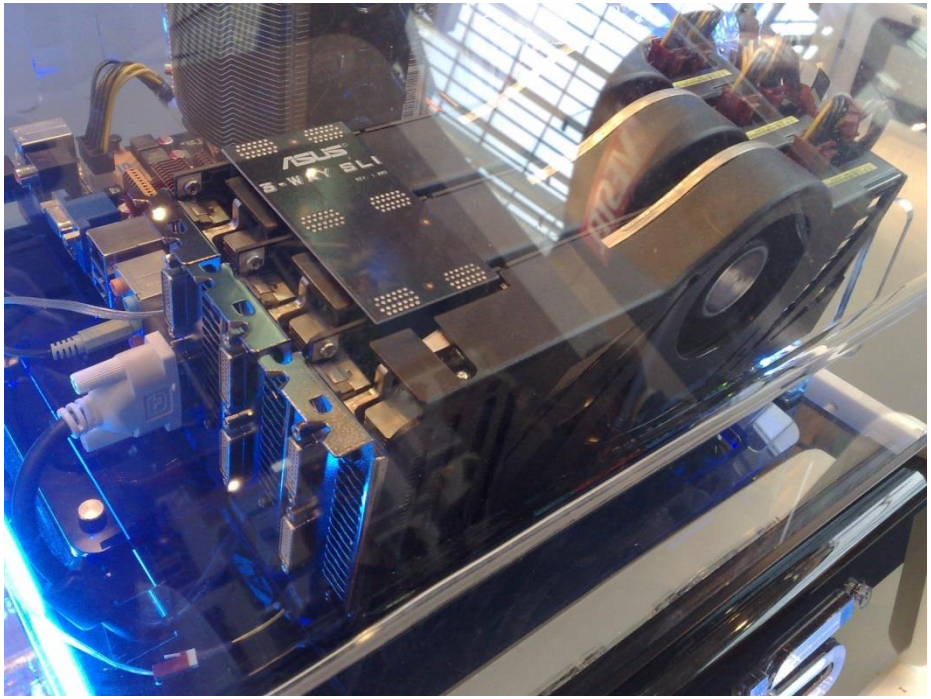
# Game Technology

Lecture 5 – 21.11.2017
Hardware Rendering

TECHNISCHE UNIVERSITÄT DARMSTADT



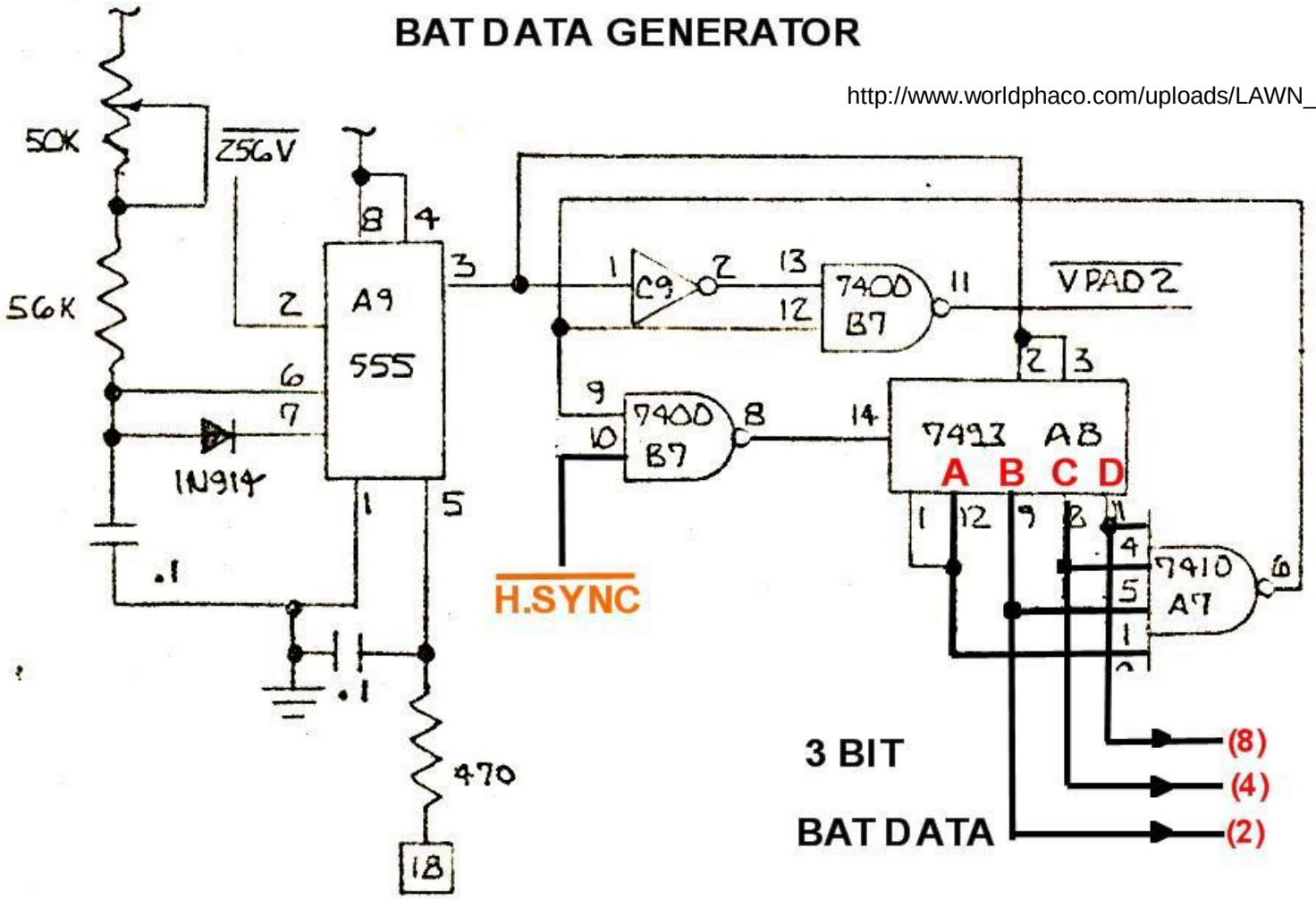

Dipl.-Inform. Robert Konrad
Polona Caserman, M.Sc.

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

# Pong & Computer Space


Pong (1972), Computer Space (1971)

# Pong “Game Engine”



# Apple 2 (1977)

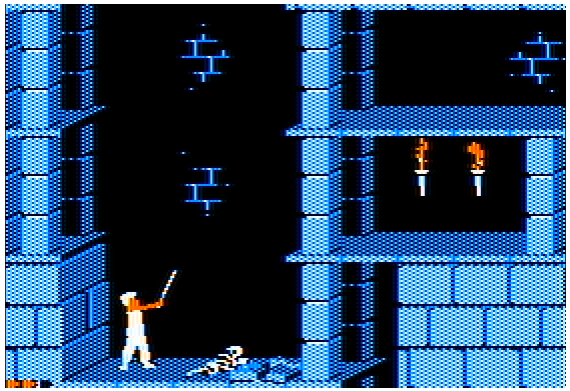
# Apple II

TECHNISCHE UNIVERSITÄT DARMSTADT

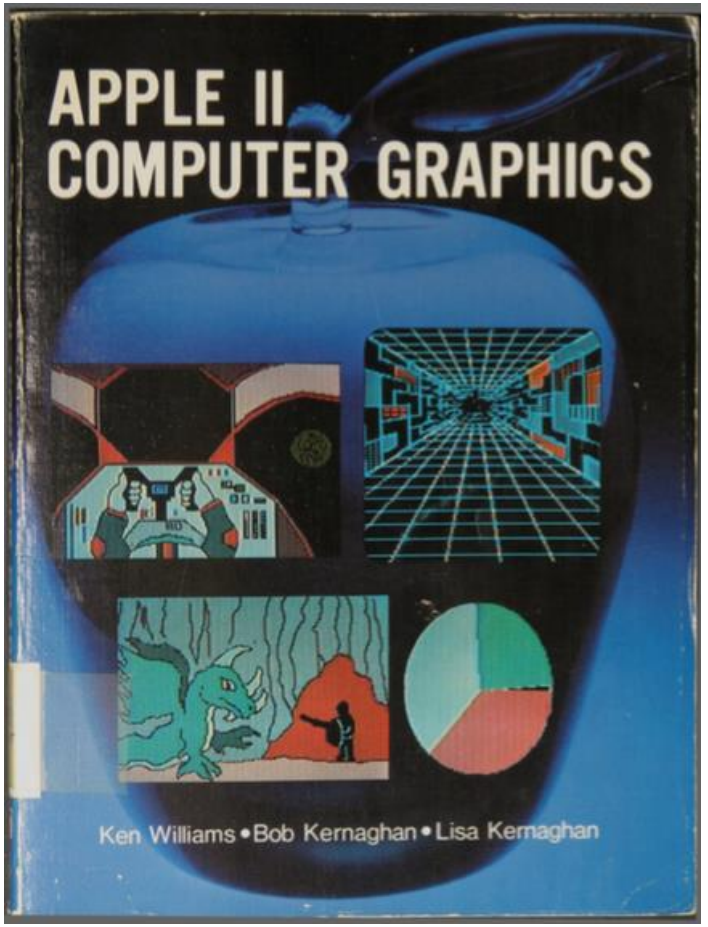
**One of the first mass-produced home computers with CG capabilities**

- MOS Technologies 6502 at 1 MHz
- 4k or more
- Actual framebuffer


Mystery House (1980)



Prince of Persia (1989)



# Atari VCS (1977)

# Atari VCS

TECHNISCHE UNIVERSITÄT DARMSTADT

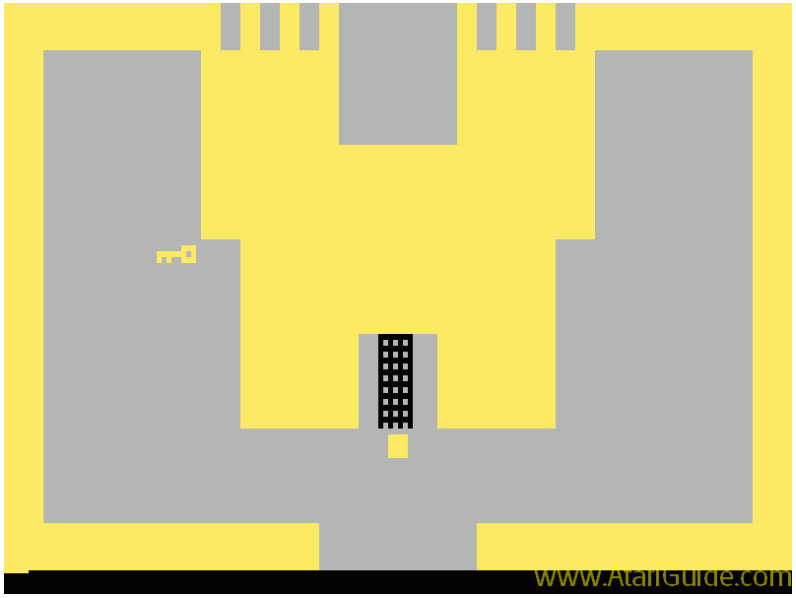
**Later renamed to Atari 2600**

**MOS Technologies 6507**

- Variant of 6502: Addressable memory reduced from 64 kB to 8 kB
- ~1,19 MHz
- 128 Bytes

**Developers had to be very creative**

- E.g. build mirrored levels
- Use the timing of the monitor to switch colors in one frame
- Use undocumented features




Adventure (1979)

**More info: “Racing the Beam: The Atari Video Computer System”**

# Nintendo Entertainment System/Famicom (1983 Japan, 1985 USA, 1986 Europe)

# Nintendo Entertainment System/Famicom

TECHNISCHE UNIVERSITÄT DARMSTADT

**CPU: Ricoh 2A03 (6502-base) @ 1,77 MHz (PAL) / 1,79 MHz (NTSC)**

**(no big difference to the VCS)**

**RAM: 2k (plus game ROM)**

**Graphics: PPU Ricoh-Chip (NTSC: RP2C02, PAL: RP2C07) @ 5,37 MHz bzw. 5,32 MHz**

**Supports sprites and tilesets**

# NES quirks

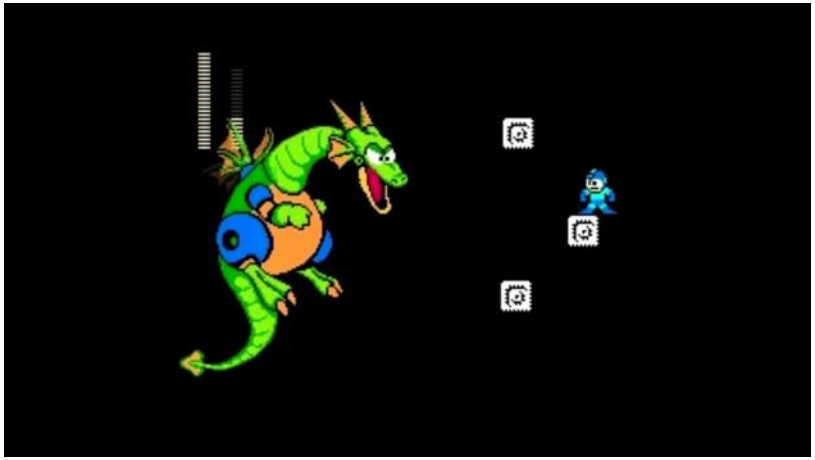
TECHNISCHE UNIVERSITÄT DARMSTADT

## Sprite flickering

- Happened when too many sprites were being drawn in one line

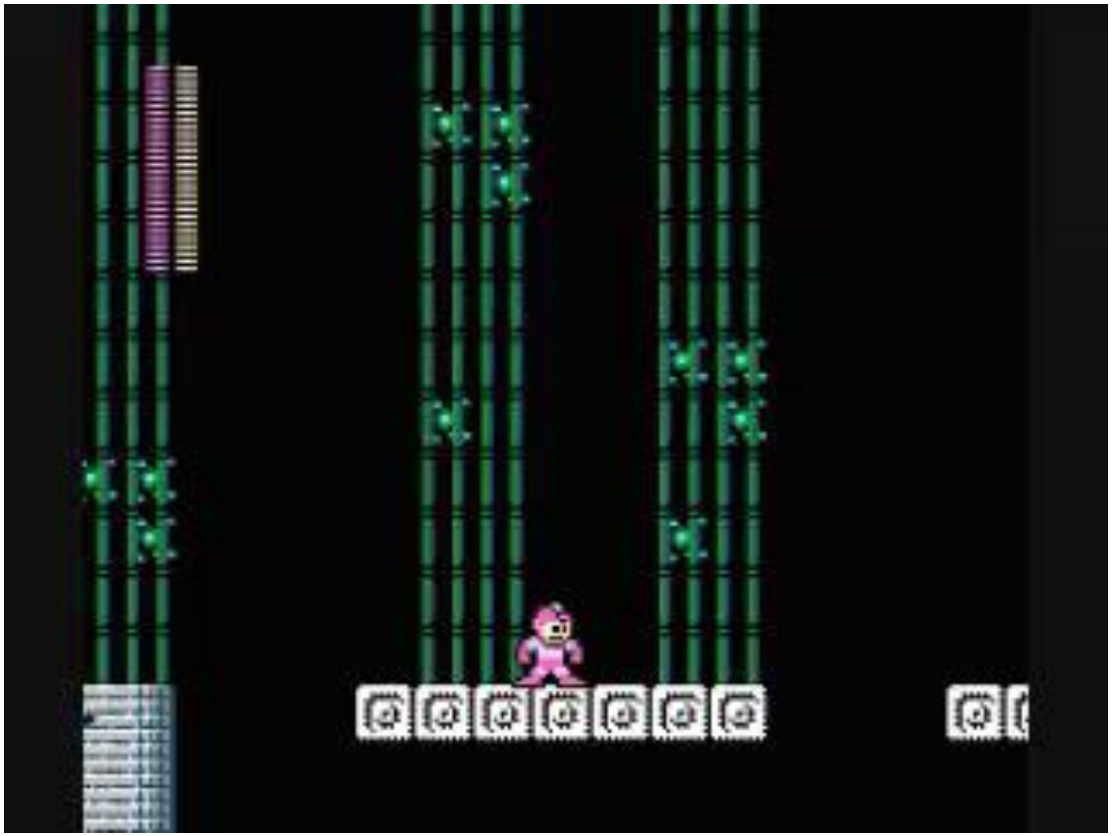
## Limited sprite size

- Boss fights typically use tiled backgrounds for bosses



Mega Man 2 (1988)

# NES Quirks



https://www.youtube.com/watch?v=JrH5Q8gssvY

# Amiga 500 (1987)

TECHNISCHE UNIVERSITÄT DARMSTADT

**Motorola 68000 @ 7 MHz**

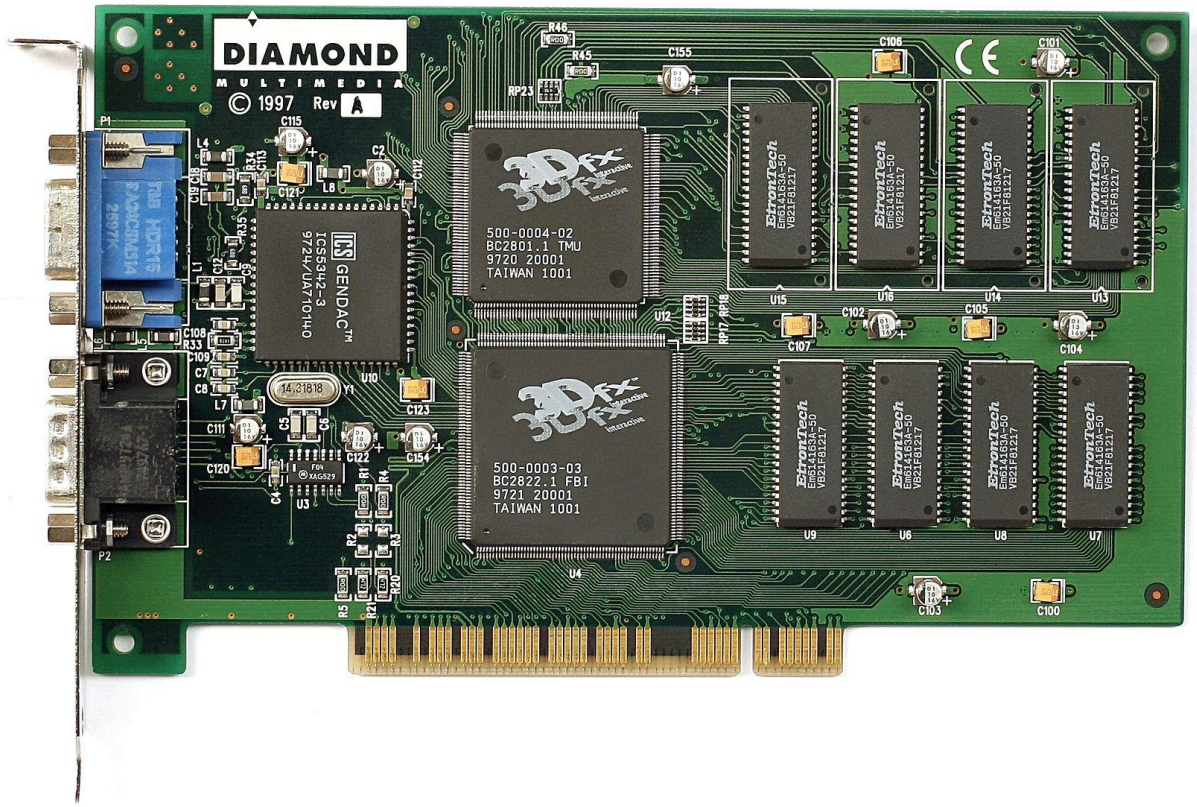
**512k**

**Programmable pixel shaders**

- One bitwise operation
- http://www.codersnotes.com/notes/command-and-vector-processors/

# IBM PC (1981)

# Voodoo Graphics (1996)



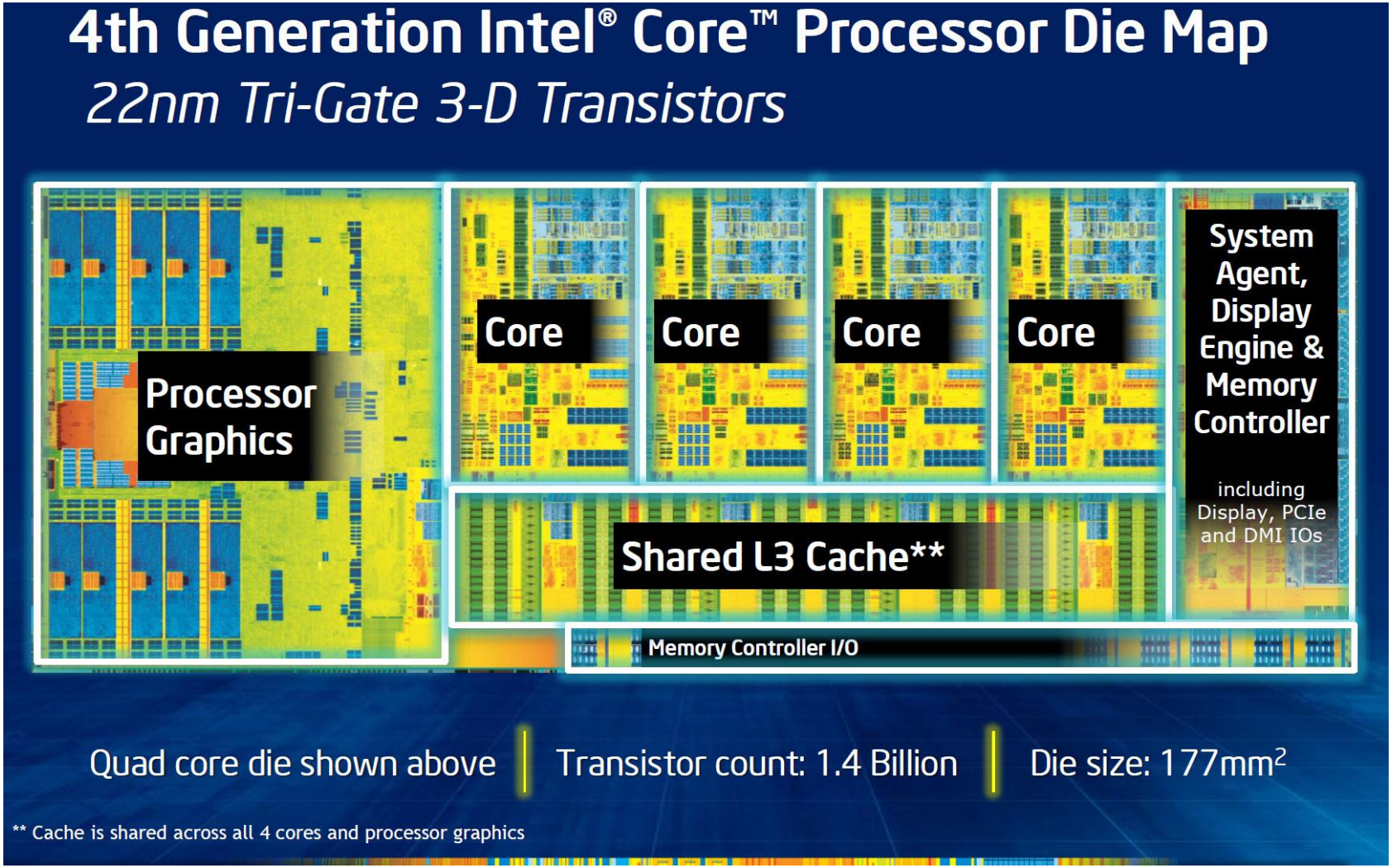
# Features of Voodoo Graphics chip

TECHNISCHE UNIVERSITÄT DARMSTADT

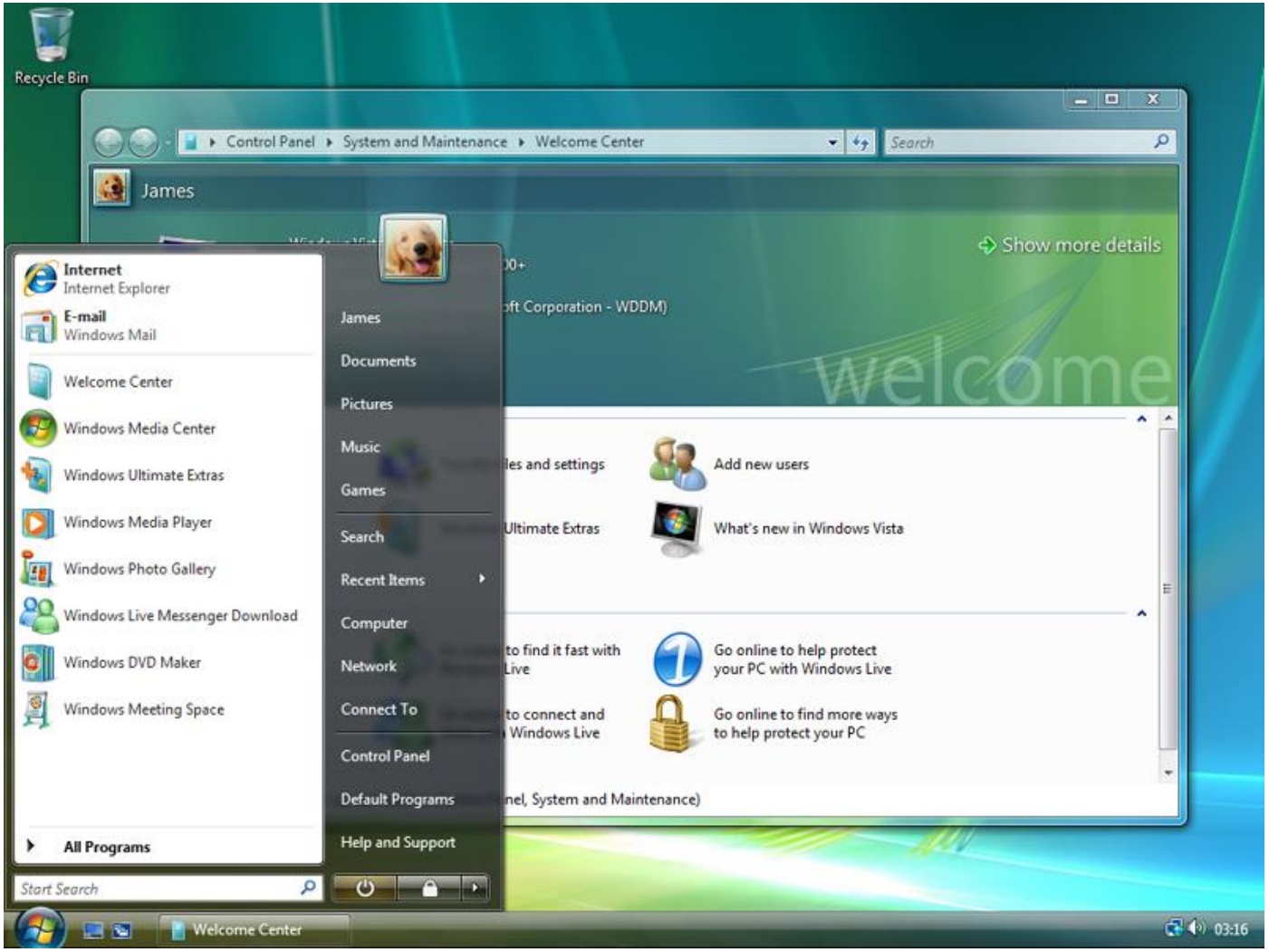
- Triangle raster engine
- Linearly interpolated Gouraud-shaded rendering
- Perspective-corrected (divide-per-pixel) texture-mapped rendering with iterated RGB modulation/addition
- Detail and Projected Texture mapping
- Linearly interpolated 16-bit Z-buffer rendering
- Perspective-corrected 16-bit floating point W-buffer rendering (patent pending)
- Texture filtering: point-sampling, bilinear, and trilinear filtering with mipmapping
- …

# Modern intel CPUs

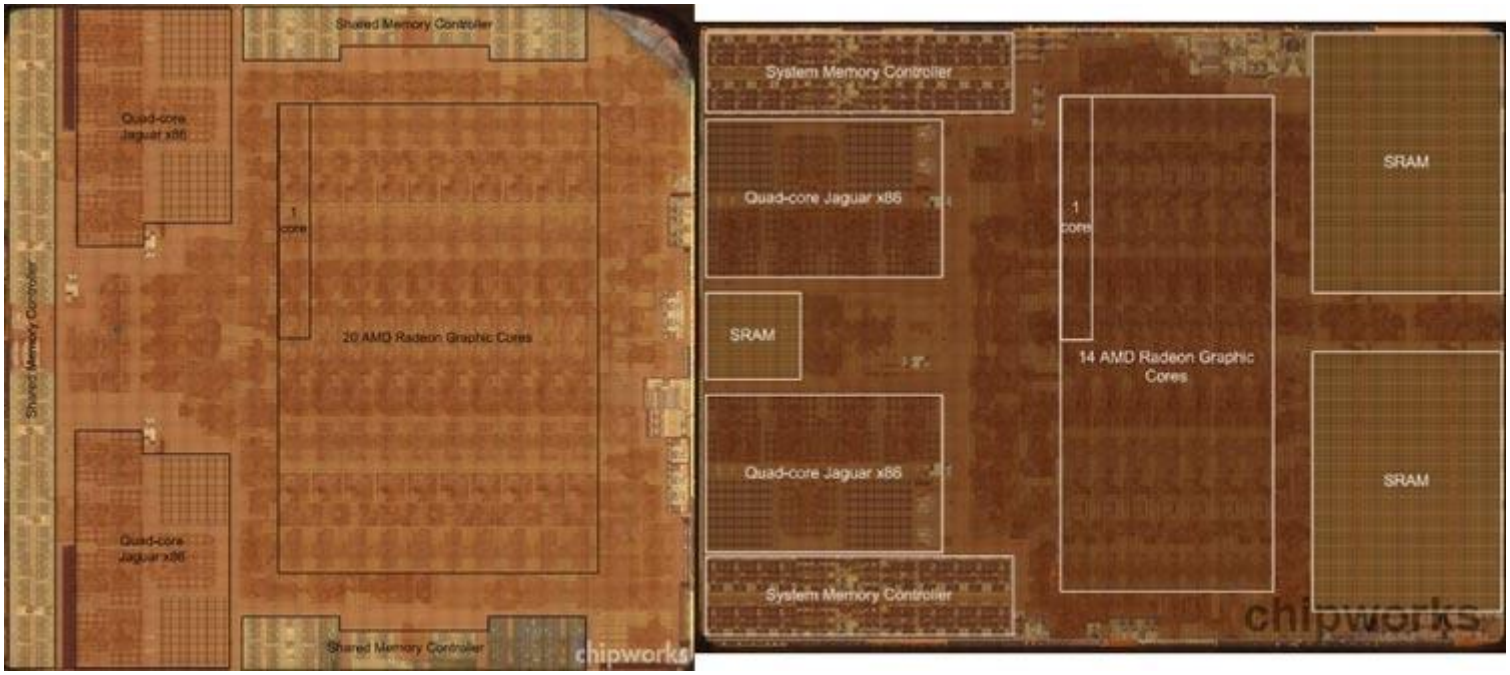
TECHNISCHE UNIVERSITÄT DARMSTADT



# Windows Vista (2007)



# Modern consoles



# CPU vs GPU

TECHNISCHE UNIVERSITÄT DARMSTADT

## CPU

- Run sequential code as fast as possible

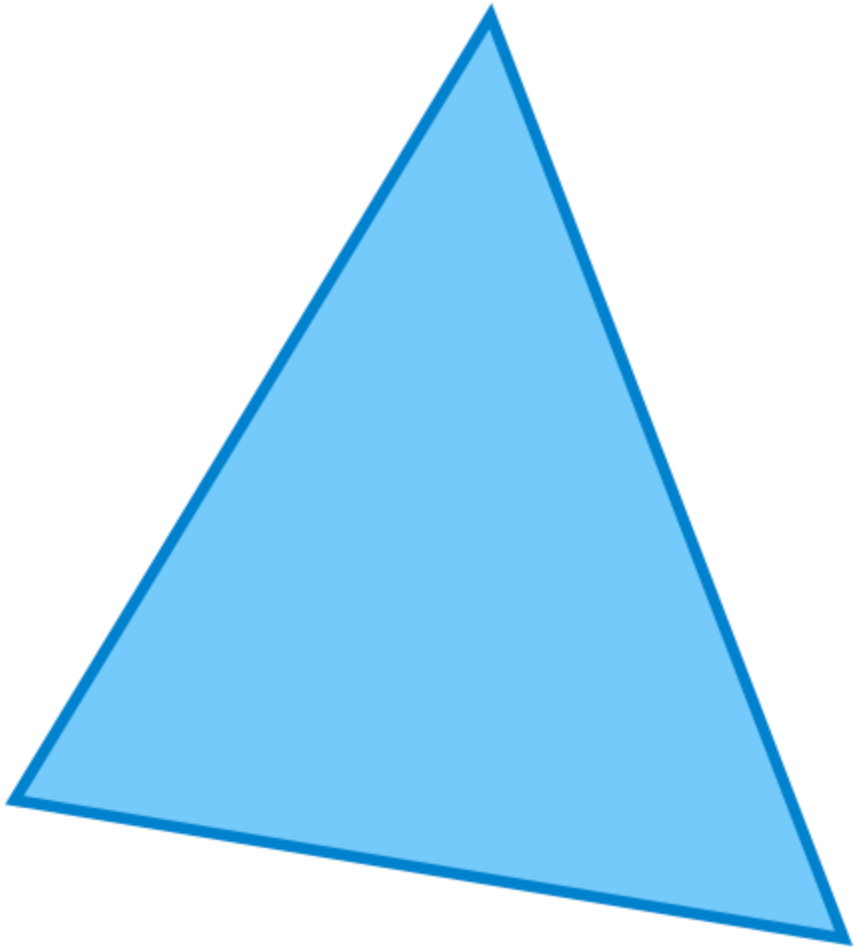
## GPU (Graphical Processing Unit)

- Massively parallel code execution
- Plus triangle rasterizer
- Plus texture sampler

## GPGPU (General purpose computations on GPU)

- Programmable compute units, not directly tied to graphics anymore
- Carry out a computation massively parallelized

# Triangles



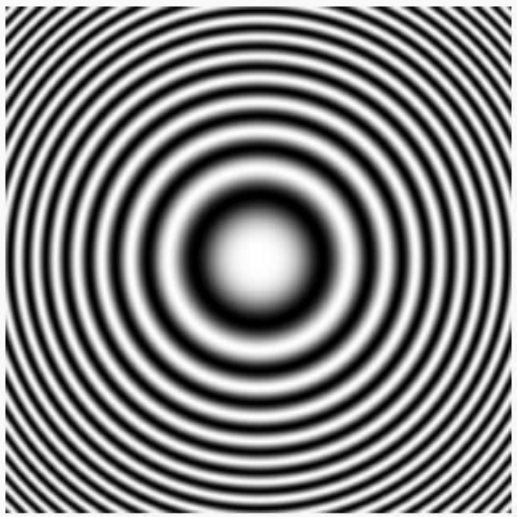
# Aliasing

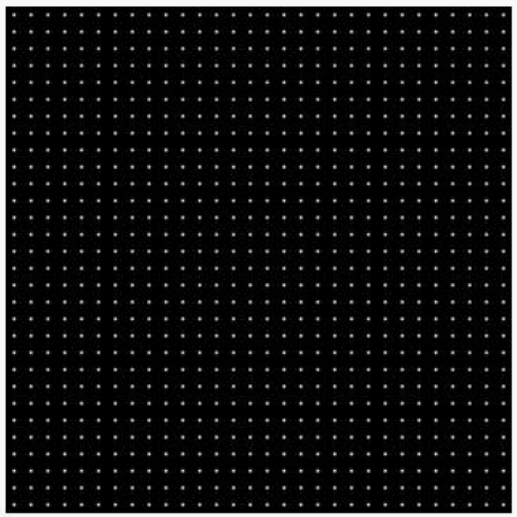
# Aliasing

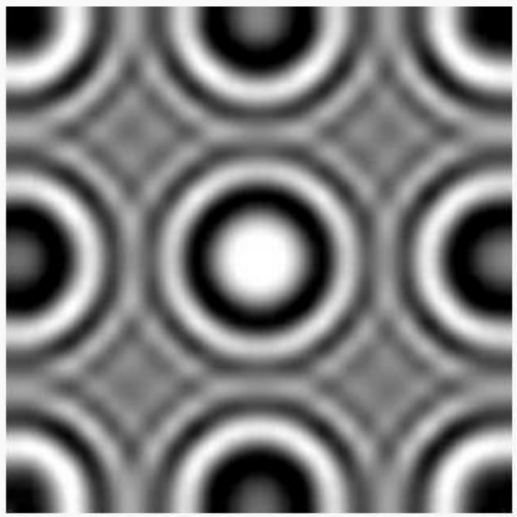
TECHNISCHE UNIVERSITÄT DARMSTADT

## Sampling frequency is too low

- Example: Original wave on the left
- Sample points in the pixel centers
- Inaccurately sampled wave on the right







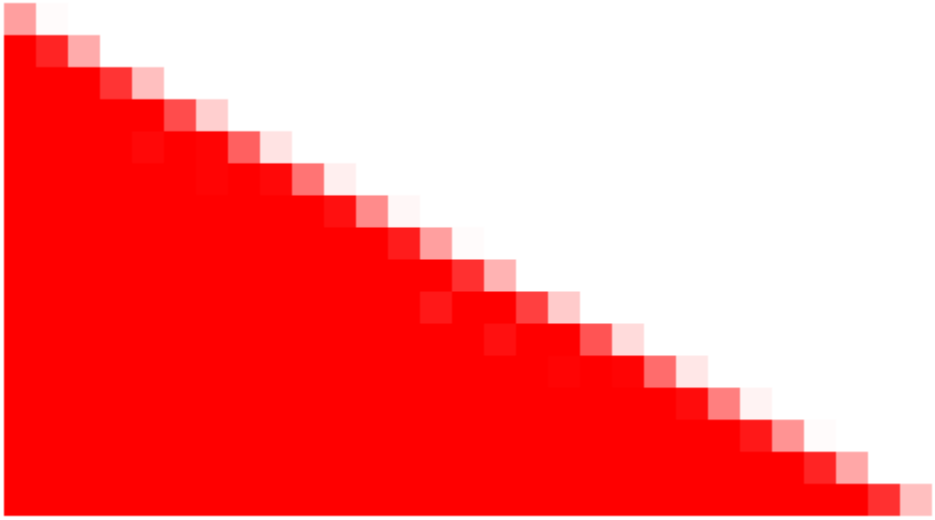
# Edge Antialiasing

TECHNISCHE UNIVERSITÄT DARMSTADT

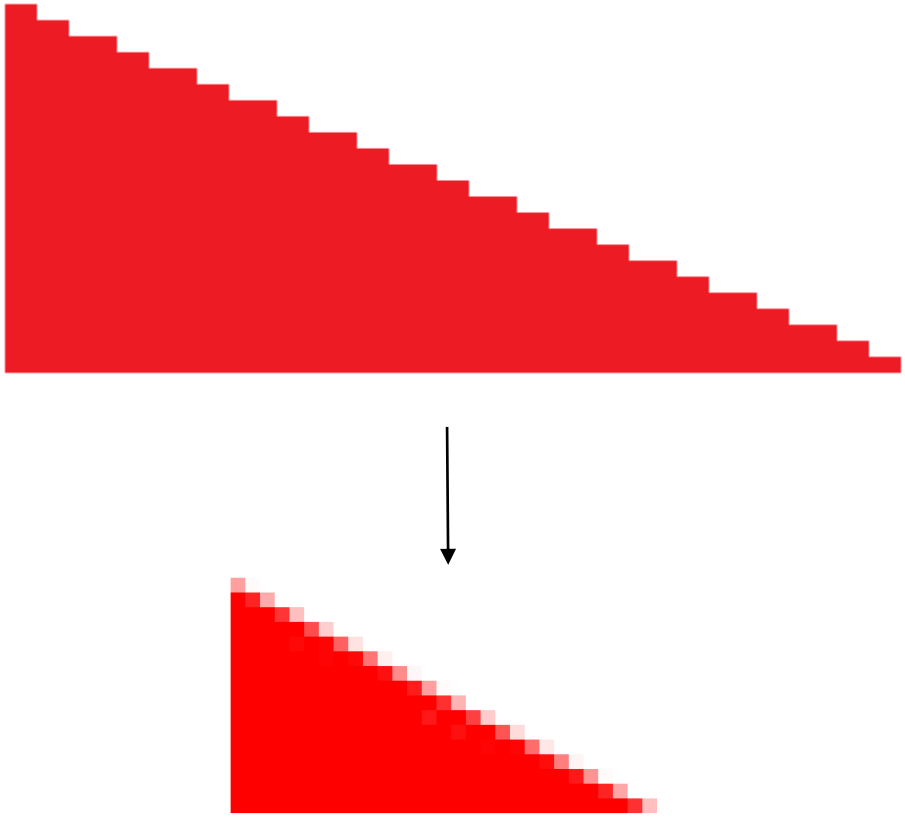
**Specifically works on edges**

**Blend with the background**

**Would require back-to-front rendering**



# Supersample/Multisample Antialiasing



# Postprocess Antialiasing

TECHNISCHE UNIVERSITÄT DARMSTADT

# Temporal Anti-Aliasing

TECHNISCHE UNIVERSITÄT DARMSTADT

**Use information from several frames for a cleaner image**

**or**

**Anti-Aliasing done over several frames, to remove effects seen during motion**

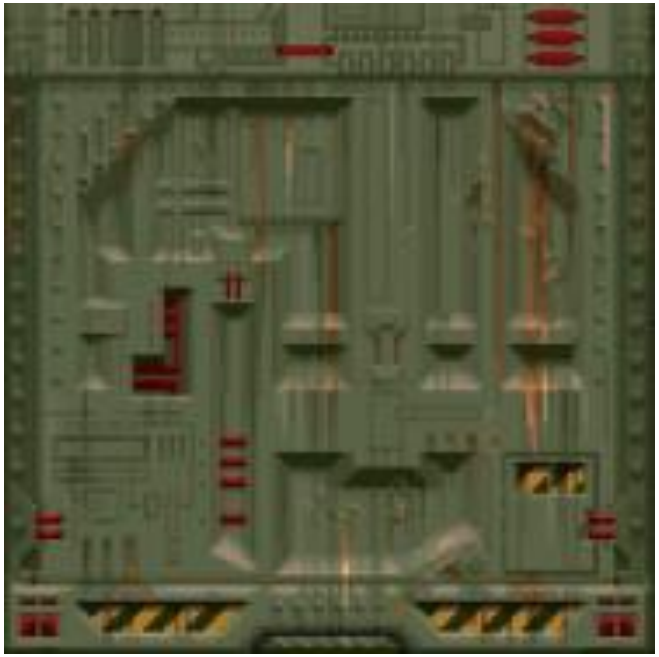
# Textures

TECHNISCHE UNIVERSITÄT DARMSTADT

**Basically images**

**Preferably $2^n * 2^n$**

- Other sizes not necessarily supported
  - Expand image and fix up texture coordinates



# Texture Sampling

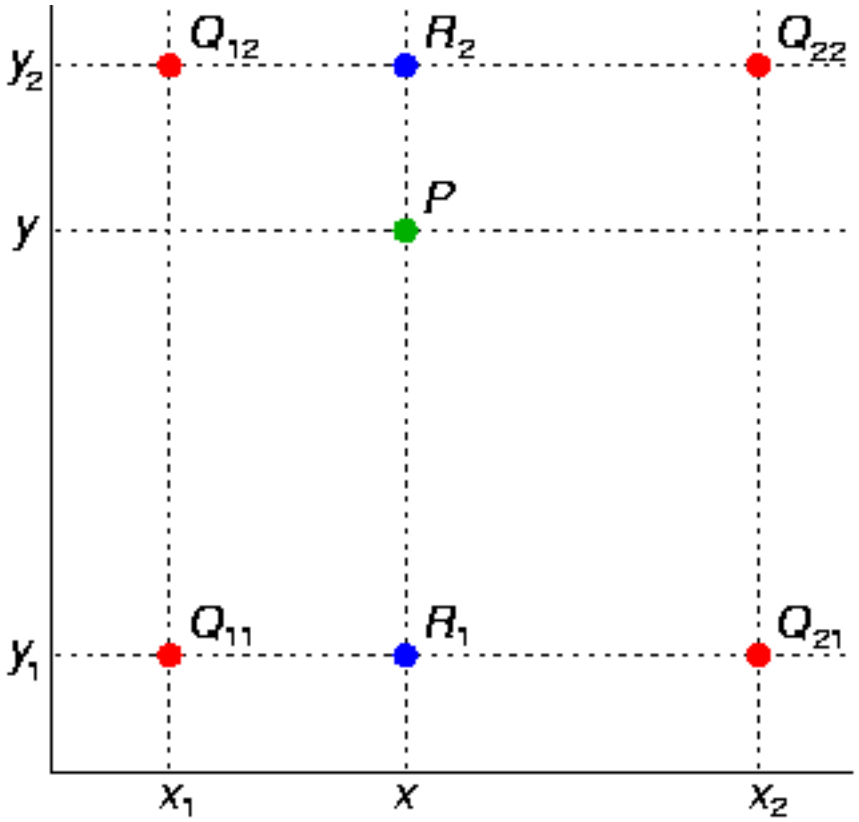
TECHNISCHE UNIVERSITÄT DARMSTADT

## Point Filtering

## Bilinear Filtering

- Interpolate four neighbouring pixels

# Bilinear filtering



# Mip Mapping

TECHNISCHE UNIVERSITÄT DARMSTADT

**Example: Texture mapped to one pixel**

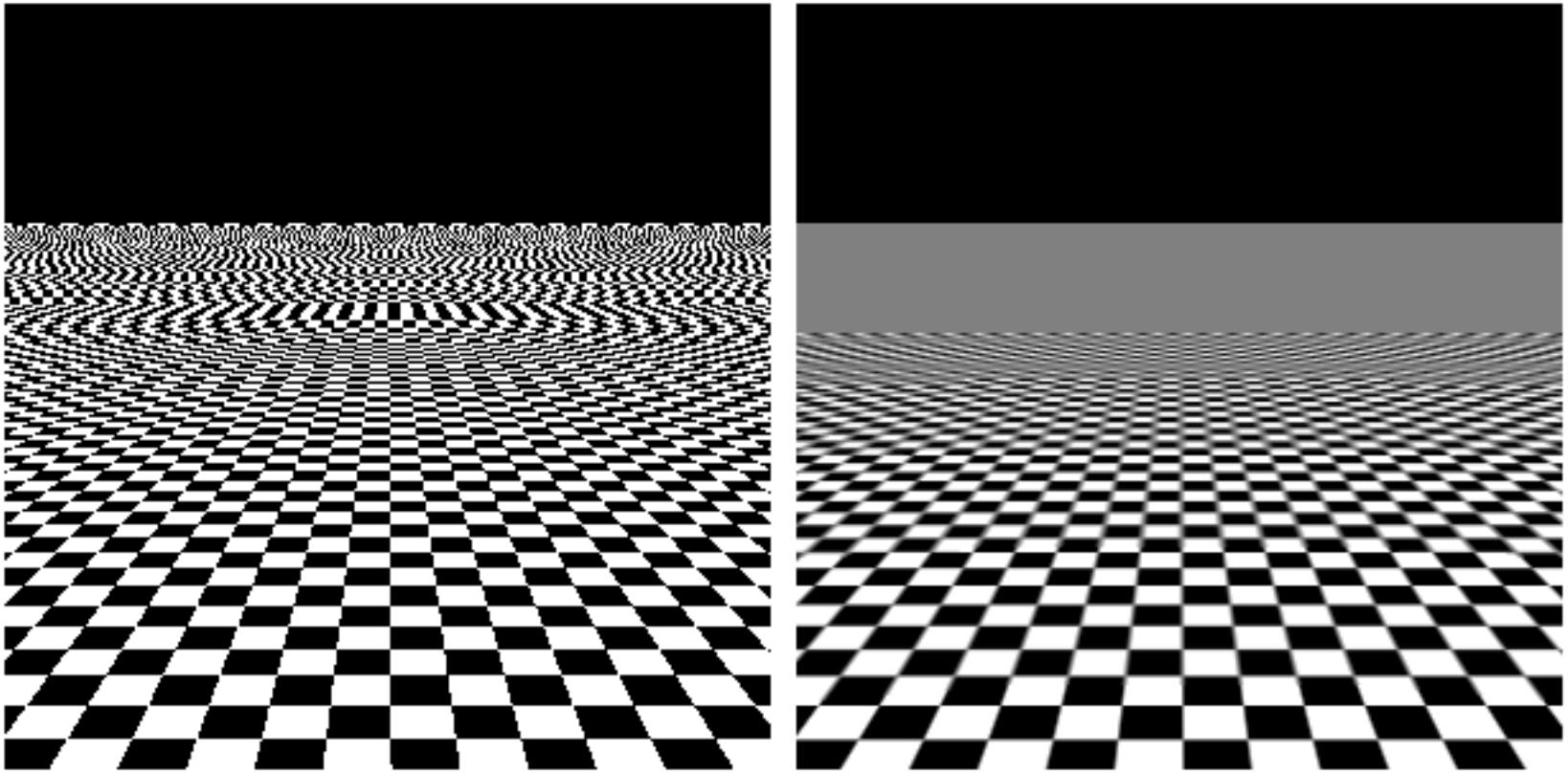
- Ideally calculate mean color value of the complete texture

**Trick: Precompute images**

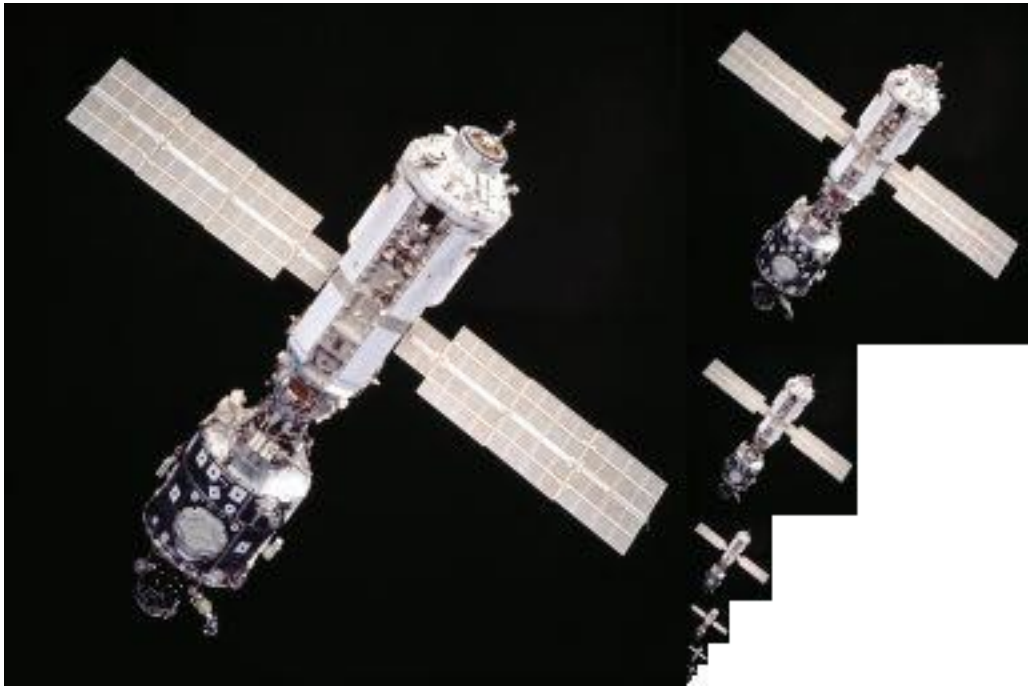
- Width / 2, Height / 2
- Width / 4, Height / 4
- …
- Sample from best fitting image

**(***multum in parvo, „much in little“)*

# No mip mapping



# MIP Mapping



# Mip Mapping

TECHNISCHE UNIVERSITÄT DARMSTADT

**Seams between mip levels are often visible**

- Trilinear filtering


**Perspective stretches images differently in x and y**

- No optimal mip level

# Anisotropic Filtering

TECHNISCHE UNIVERSITÄT DARMSTADT

# Anisotropic filtering

# Depth Buffer



**Implemented in hardware**

**Used automatically by the rasterizer**

**3D APIs offer simple configuration**

- Off, allow only smaller values, allow only larger values

# Alpha-Blending

TECHNISCHE UNIVERSITÄT DARMSTADT

## Critical for performance

- Reads in previous pixels, stresses memory interface
- Makes parallel execution more difficult

## Fixed modes

- 1 * new pixel + 0 * old pixel
- source alpha * new pixel + (1 - source alpha) * old pixel
- …
- (destination alpha is rarely used)

# Programmable Blending

TECHNISCHE UNIVERSITÄT DARMSTADT

**Render to texture**

**Draw rendered texture**

**Draw blended geometry**

- Use rendered texture as input

**Much slower**

**Unless GL_EXT_shader_framebuffer_fetch**

# Most used blending modes

TECHNISCHE UNIVERSITÄT DARMSTADT

## Standard blending

- source alpha * new pixel + (1 - source alpha) * old pixel

## Additive blending

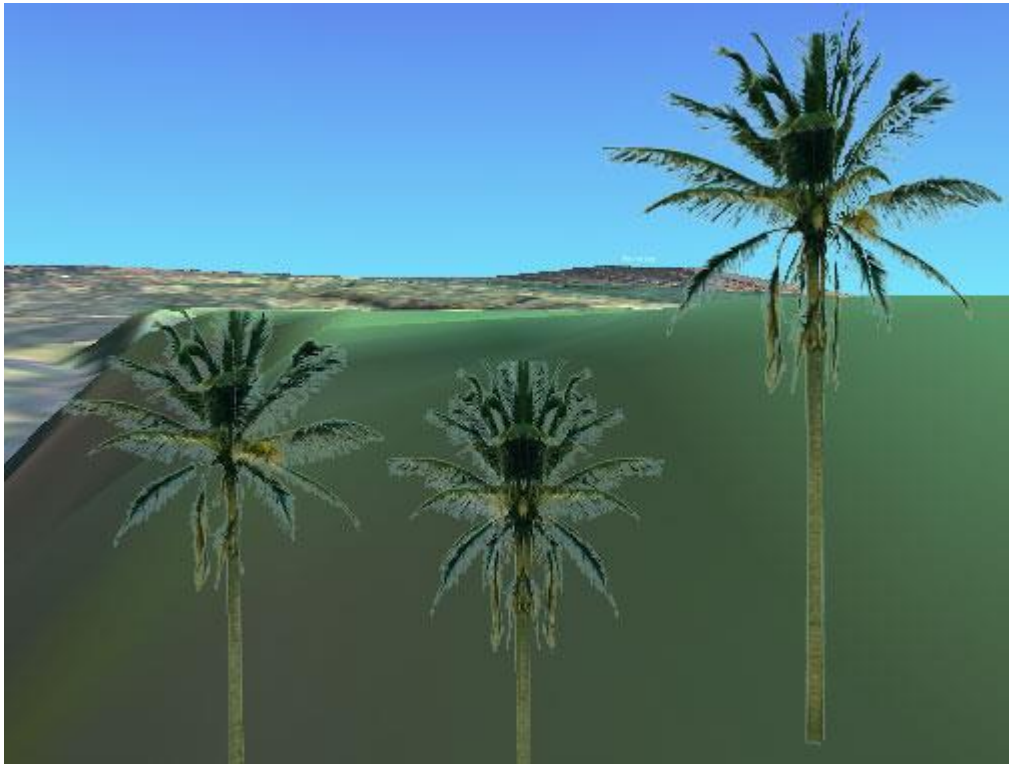
- source alpha * new pixel + old pixel

# Texture Sampling and Transparency

TECHNISCHE UNIVERSITÄT DARMSTADT

**Bilinear filtering samples rgb + alpha**

**At alpha borders samples rgb values with alpha 0**



# Premultiplied Alpha

TECHNISCHE UNIVERSITÄT DARMSTADT

**Multiply rgb with alpha**

**Fixes texture sampling (invisible pixels are multiplied with 0)**

**Fixes sunglasses**

- Premultiply alpha, then add something
- Combines standard and additive blending

**Blending mode:**

- new pixel + (1 - source alpha) * old pixel

# Vertex Shader

TECHNISCHE UNIVERSITÄT DARMSTADT

**Calculates vertex transformations**

**Prepares additional data for later shader stages**

**→ What we did in Exercise 3**

# Fragment Shader

TECHNISCHE UNIVERSITÄT DARMSTADT

**Also referred to as Pixel Shader**

**Uses interpolated data from vertex shader**

**Calculates colors**

**→ What we did in Exercise 4**

# Vertex Buffer

TECHNISCHE UNIVERSITÄT DARMSTADT

**Array of vertices**

**Can hold additional data per vertex**

- E.g normal, animation data, ...

**Has to assign additional data to names or registers for vertex shader**

**Primary interface from CPU to GPU**

# Index Buffer

TECHNISCHE UNIVERSITÄT DARMSTADT

**Array of indices**

**That‘s it**

**→ One vertex can be re-used in several triangles**

# Draw Calls

TECHNISCHE UNIVERSITÄT DARMSTADT

**Set Vertex Shader**

**Set Fragment Shader**

**Set IndexBuffer**

**Set Vertex Buffer**

**DrawIndexedTriangles()**

**DrawIndexedTriangles()**

**…**

# Implicit Work



**Create command buffers**

**Verify data (?)**

**Memory layouts and locations**

**Compile shaders**

**Synchronize resources (draw calls when vertex buffer still in use?)**

# Compute Shader → GPGPU

TECHNISCHE UNIVERSITÄT DARMSTADT

**No Rasterization**

**Define work group sizes manually**

**Many competing languages**

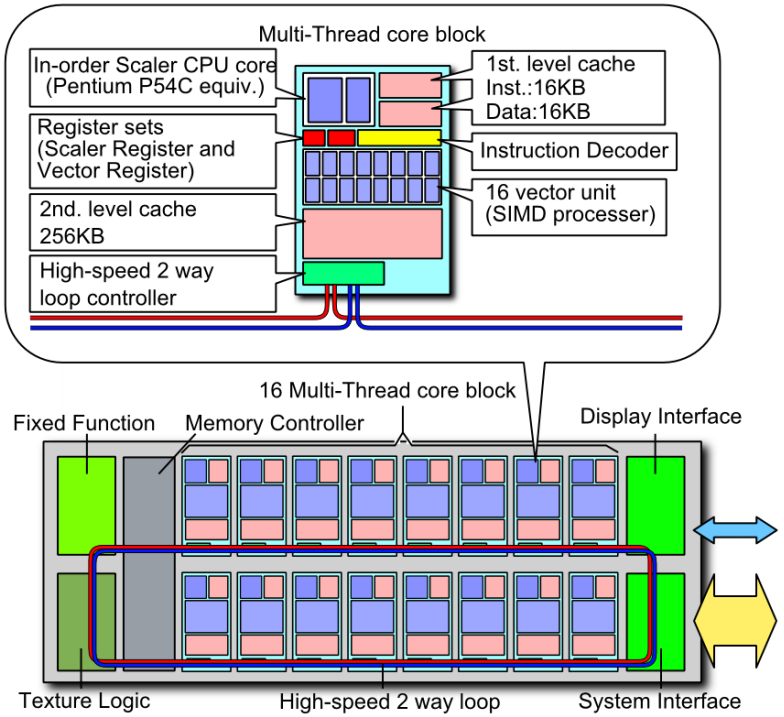
- Even OpenCL and GLSL compute shaders

# Triangles on Compute

TECHNISCHE UNIVERSITÄT DARMSTADT

## Xeon Phi

- Ex project Larrabee




- https://code.google.com/p/cudaraster/
  - From nVidia

# More Shaders

TECHNISCHE UNIVERSITÄT DARMSTADT

## Geometry Shader

- Works on complete triangles
- Can often be replaced with instanced rendering

## Tessellation Shader

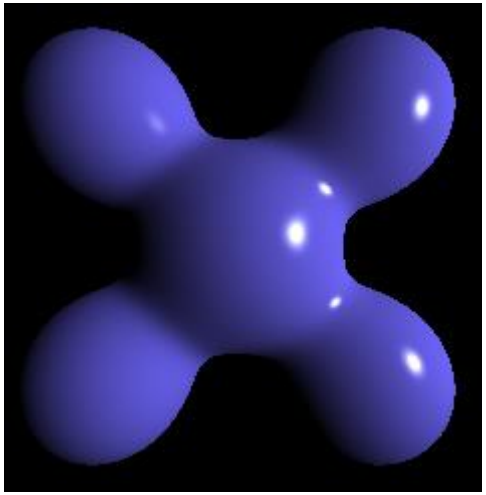
- Can create new triangles

# Phong Lighting

TECHNISCHE UNIVERSITÄT DARMSTADT

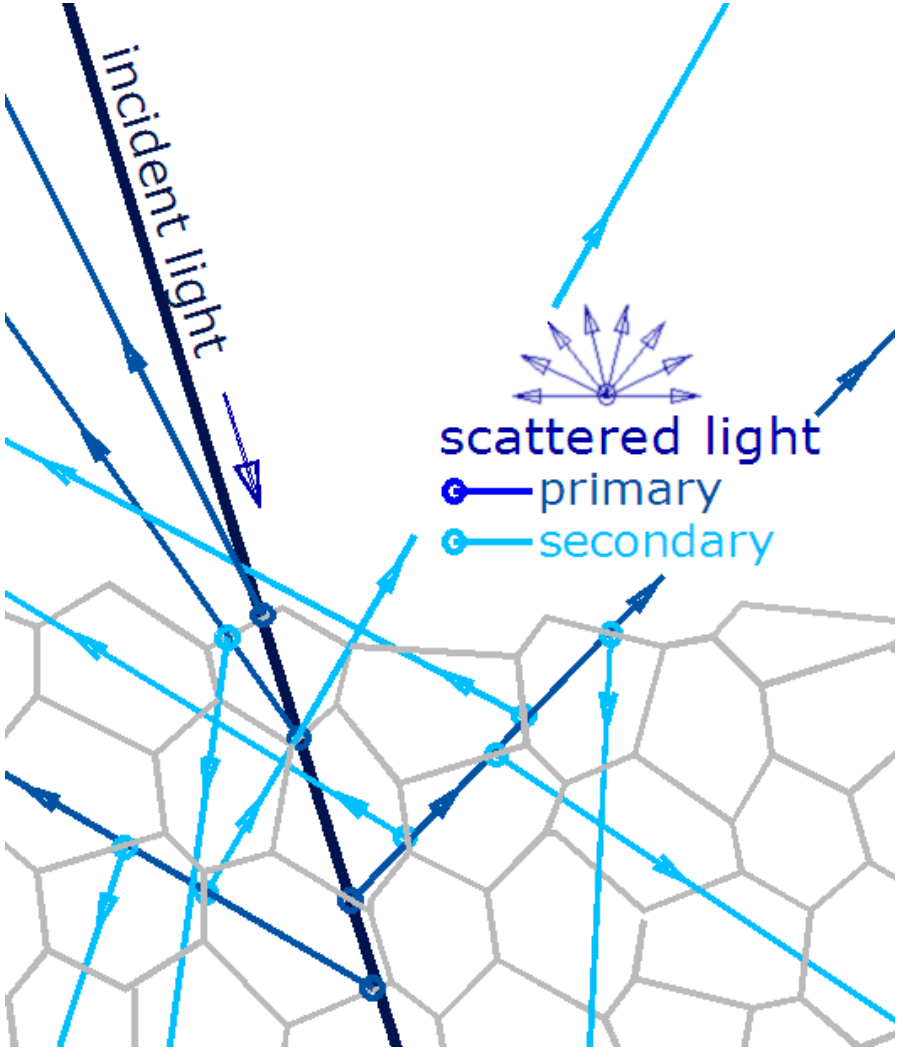
**color = ambient + diffuse + specular**

- Note: Light from different sources can always be added just like that



# Ambient = Constant

# Diffuse

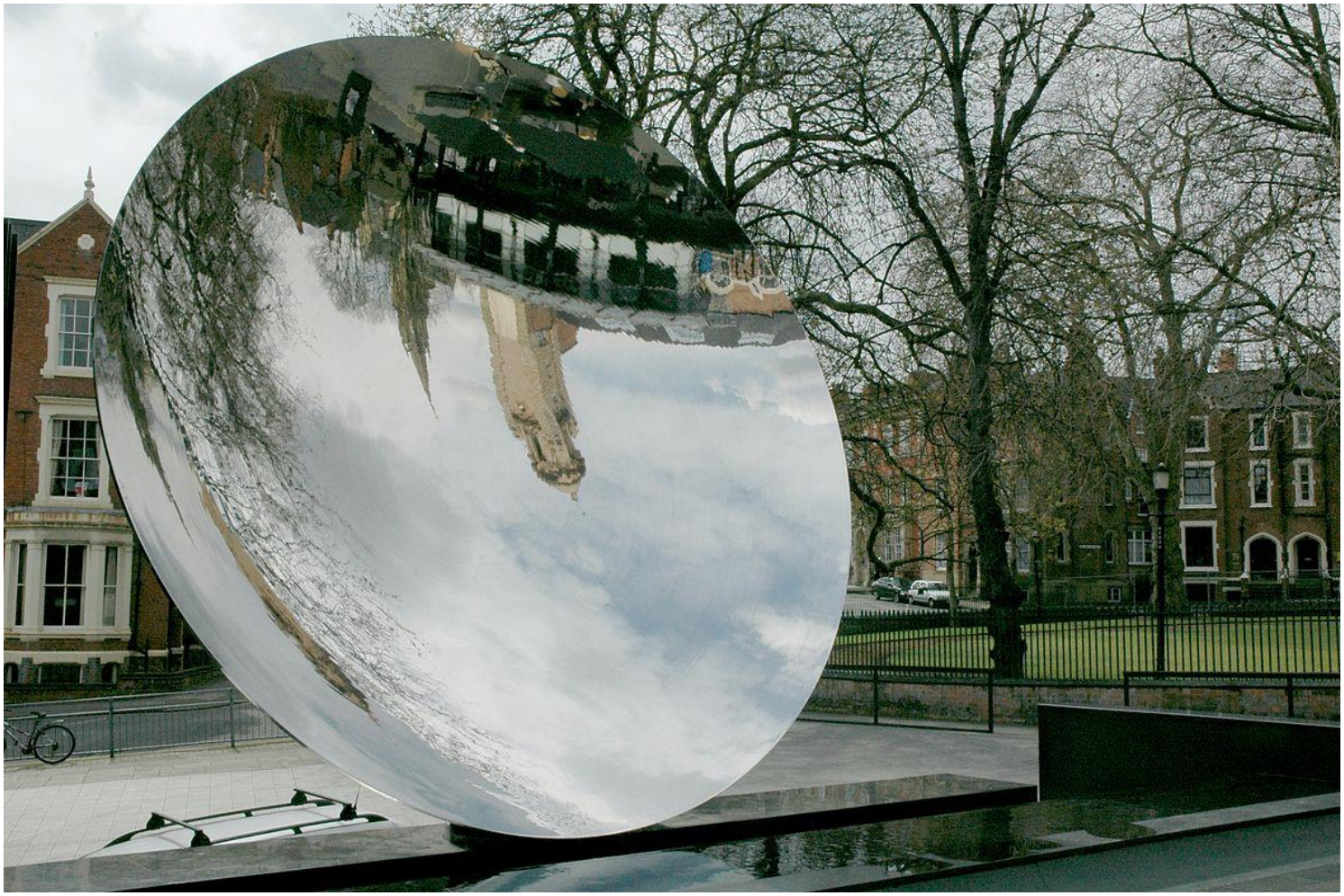


# Diffuse

TECHNISCHE UNIVERSITÄT DARMSTADT

**diffuse = LN (see previous lecture)**

# Specular



# Specular

$$I_{specular} = I_{in} k_{specular} cos^{n} \theta$$

$$I_{specular} = I_{in} k_{specular} \left(\vec{R} \cdot \vec{V}\right)^{n}$$

**R: mirrored vector to the light source (reflectance vector)**

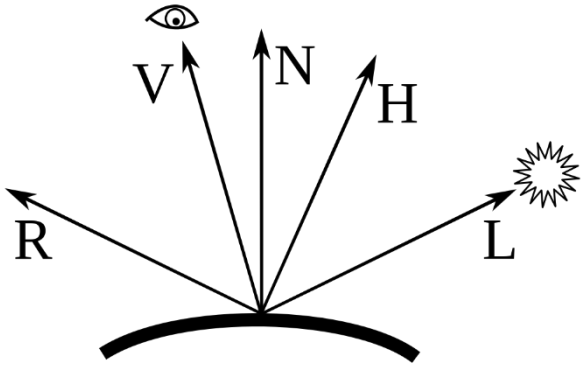
**V: vector to the camera**

**n: shininess – start at 32 and tune**

**k: empiric reflection factor**

**Empirical model**

**Ugly for larger angles (cos → 0)**




**(H: Half-vector between V and L)**

**(N: Normal)**

# Blinn Phong

TECHNISCHE UNIVERSITÄT DARMSTADT

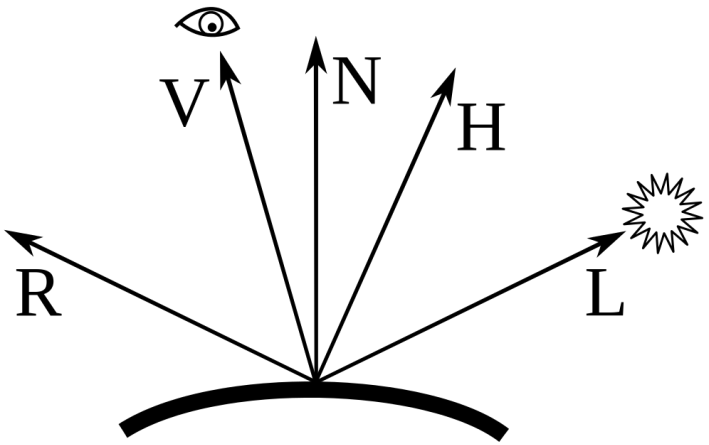
$$\boldsymbol{H = \frac{V + L}{\|V + L\|}}$$

$$\boldsymbol{I_{specular} = I_{in} k_{specular} cos^{n} \theta'}$$

$$\boldsymbol{I_{specular} = I_{in} k_{specular} \cdot \left( \frac{(V + L) \cdot N}{\|(V + L)\| \cdot \|N\|} \right)^{n}}$$

**A little faster**

**A little nicer**



# Better ambient light

TECHNISCHE UNIVERSITÄT DARMSTADT

## Real ambient light is hard

- Light bouncing and bouncing and bouncing…

## Ambient light tends to look very diffuse

- No hard borders

## Precompute everything

- Put it in small textures
- Bilinear filtering blurry stuff works wonderfully

# Light Baking


Quake (1996)

# Better specular lighting

TECHNISCHE UNIVERSITÄT DARMSTADT

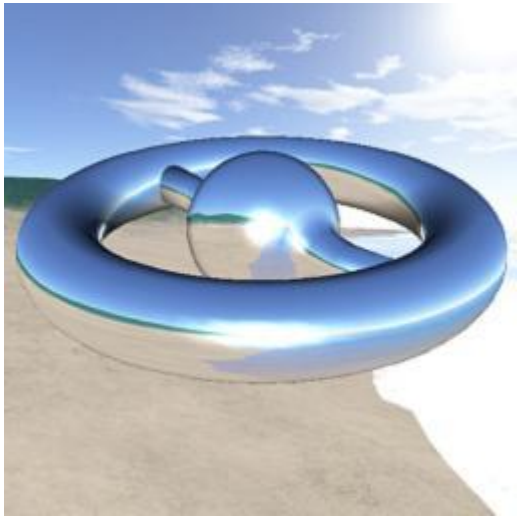
**Render six orthogonal perspectives into a cube map**

- Camera center = center of object to be rendered

**Sample vector into cubemap for every pixel**

**Obviously very expensive**

**Can not be precomputed**



# Ambient, Diffuse…

TECHNISCHE UNIVERSITÄT DARMSTADT

**Thinking of „Ambient“ is only an approximation**

- Phong lighting is an approximation of an approximation

**Light bounces around**

- First bounce → direct lighting (use diffuse and specular)
- Second bounce → hard shadows
- More bounces → ambient light

# Shadow Mapping

TECHNISCHE UNIVERSITÄT DARMSTADT

**Set camera to light source**

**Render depth → each pixel value = distance from light**

**During regular rendering**

**Transform vertices two times**

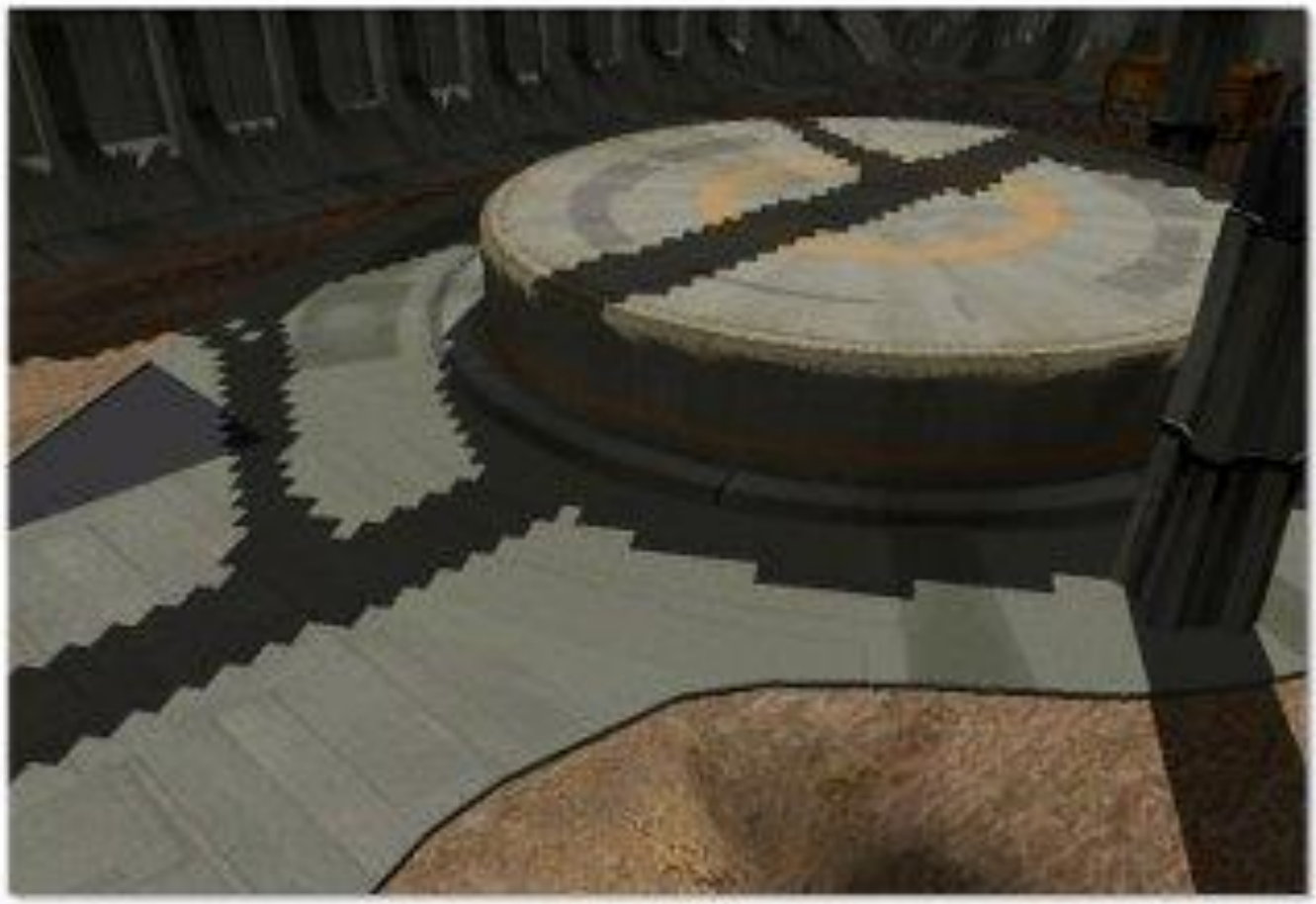
- Using camera position
- Using light position → z = distance from light

**Read depth texture**

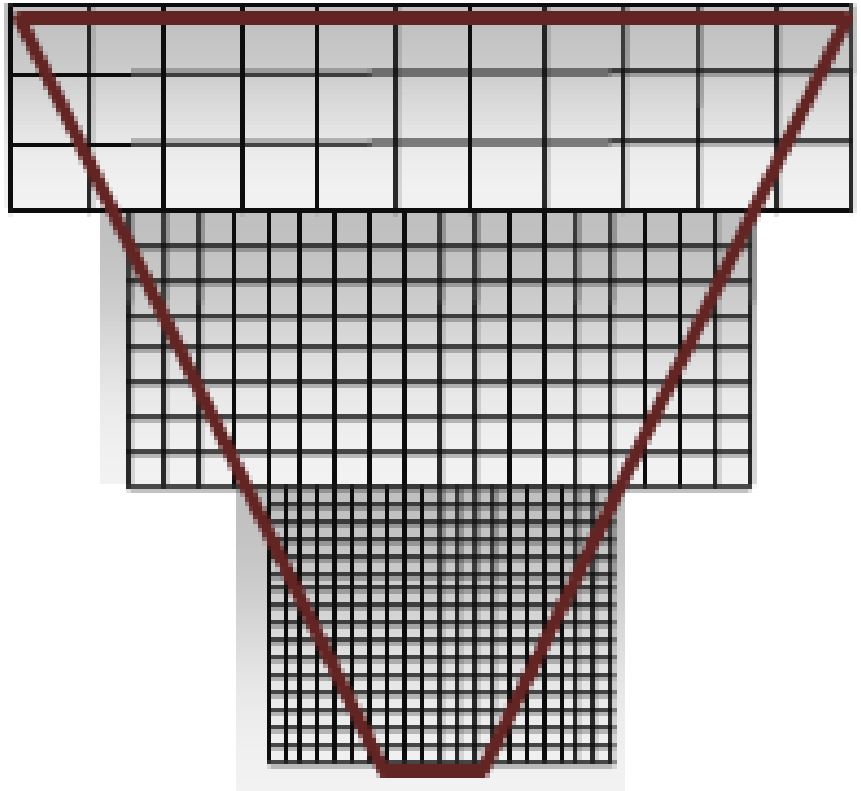
**Compare depth calculated using light pos and depth from texture**

- If greater → in shadow

# Shadow Mapping Problems



# Cascaded Shadow Maps



# Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

## What work can the GPU assist us with?

- Highly parallel calculations:
  - Graphics (each pixel, each vertex, ...)
  - General purpose tasks that can be parallelized
- Graphics-related tasks
  - Rasterization
  - Texture lookups/filtering

## Techniques

- Antialiasing
- Mip-mapping
- ...

# GLSL

TECHNISCHE UNIVERSITÄT DARMSTADT

**OpenGL Shading Language**

**Added to OpenGL in 2004 with OpenGL 2.0**
**Version 1.10**

**Similar to C**

**Semiautomatic parallelization**

# GLSL Example

TECHNISCHE UNIVERSITÄT DARMSTADT

```
uniform sampler2D tex;
in vec2 texCoord;
in vec4 color;
out vec4 frag;

void main() {
  vec4 texcolor = texture(tex, texCoord) * color;
  texcolor.rgb *= color.a;
  frag = texcolor;
}
```

# Vertex Shader

TECHNISCHE UNIVERSITÄT DARMSTADT

**Transforms vertices**

**Writes transformed vertex to a special var**

- gl_Position

**Can write additional data**

# Fragment Shader



**Writes final color to a single out value**

**Can write additional data**

- multi target rendering, gl_FragDepth,…

# Parallelism

TECHNISCHE UNIVERSITÄT DARMSTADT

**Vertex shader defines one function..**

- ...which is applied to lots of vertices in parallel

**Fragment shader defines one function...**

- ...which is applied to lots of pixels in parallel

**Programming model allows hardware to parallelize automatically**

- To multiple compute cores, SIMD units or combinations of both

# Uniforms

TECHNISCHE UNIVERSITÄT DARMSTADT

## Constants

- Do not change while shader executes
- Can be changed between draw calls

**uniform mat4 projectionMatrix;**

**uniform sampler2D tex;**

# Vertex shader ins

TECHNISCHE UNIVERSITÄT DARMSTADT

**As many as you want (in theory)**

```
in vec3 vertexPosition;
in vec2 texPosition;
in vec4 vertexColor;
```

# Vertex shader outs

TECHNISCHE UNIVERSITÄT DARMSTADT

**Transfer data to the fragment shader**

**Vertex shader → Interpolation → Fragment shader**

**Output in vertex shader = input in fragment shader**

**out vec2 texCoord;**

# Vector types

TECHNISCHE UNIVERSITÄT DARMSTADT

**vec3 position;**

**vec4 color;**

**Support basic arithmetic**

**Support swizzling**

- color.bgr
- position.xy

# Matrix types

TECHNISCHE UNIVERSITÄT DARMSTADT

**mat4 projection;**

**Supports arithmetic with vectors**

# Samplers

TECHNISCHE UNIVERSITÄT DARMSTADT

**To read textures**

```
uniform sampler2D tex;

vec4 texcolor = texture(tex, texCoord);
```

# Special vars



**gl_Position**

**gl_FragDepth**

https://www.opengl.org/wiki/Built-in_Variable_(GLSL)

- There are many more

# Precision modifiers

TECHNISCHE UNIVERSITÄT DARMSTADT

**precision mediump float;**

**Precision can be reduced**

- Often makes sense in the fragment shader
- And is often necessary (OpenGL ES)

# GLSL versions

TECHNISCHE UNIVERSITÄT DARMSTADT

**Up to version 4.6**

**Different versions for OpenGL ES**

# Kore Graphics

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#include <Kore/Graphics4/Graphics.h>
```

**Set uniforms ala**

```
ConstantLocation loc = program->getConstantLocation(„bla“);
Graphics::setFloat(loc, 2.0f);
```

**Coordinate system is (-1 to 1, -1 to 1, -1 to 1) like in OpenGL**

# Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

**OpenGL Shading Language**

**Types of shaders**

**Input and Output**

**Operations**

***More info: „Orange Book“ (OpenGL Shading Language)***