Game Technology

Lecture 6 – 21.11.2014

Dipl-Inf. Robert Konrad
Dr.-Ing. Florian Mehm

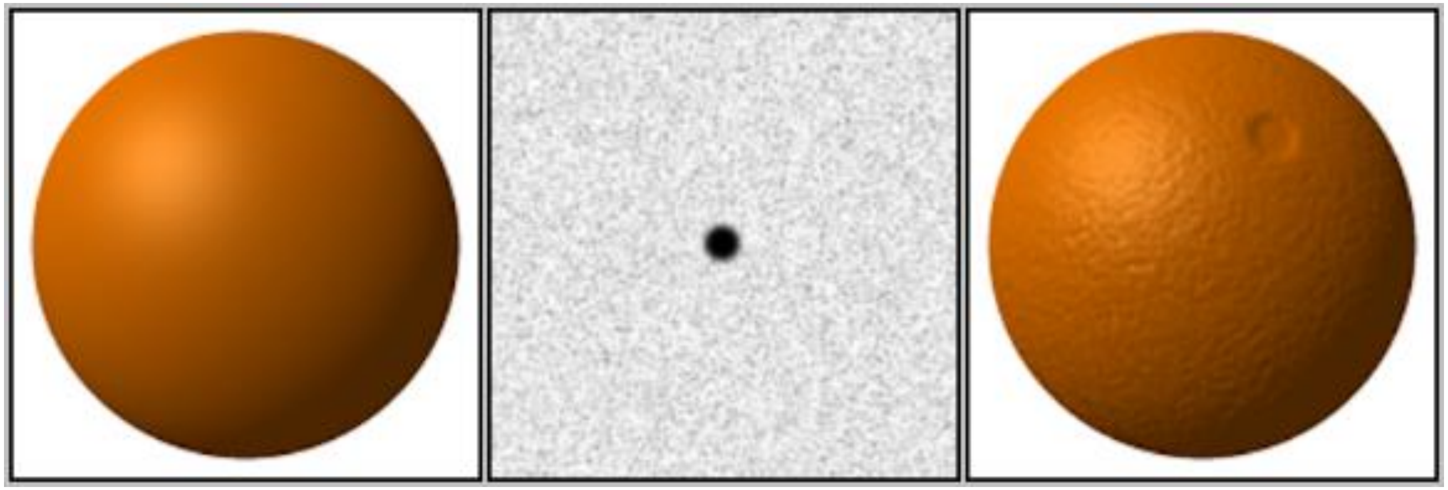
Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

Preliminary timetable

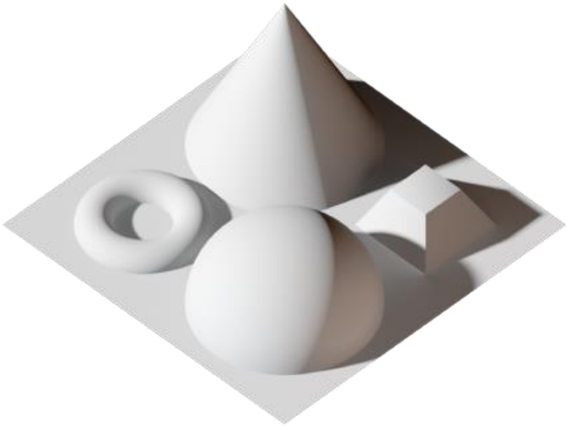
TECHNISCHE UNIVERSITÄT DARMSTADT

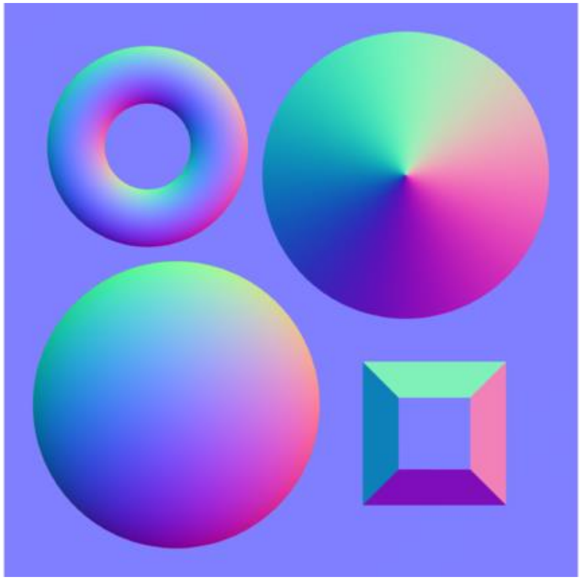
Lecture No.	Date	Topic
1	17.10.2014	Basic Input & Output
2	24.10.2014	Timing & Basic Game Mechanics
3	31.10.2014	Software Rendering 1
4	07.11.2014	Software Rendering 2
5	14.11.2014	Basic Hardware Rendering
6	**21.11.2014**	**Animations**
7	28.11.2014	Physically-based Rendering
8	05.12.2014	Physics 1
9	12.12.2014	Physics 2
10	19.12.2014	Scripting
11	16.01.2015	Compression & Streaming
12	23.01.2015	Multiplayer
13	30.01.2015	Audio
14	06.02.2015	Procedural Content Generation
15	13.02.2015	AI

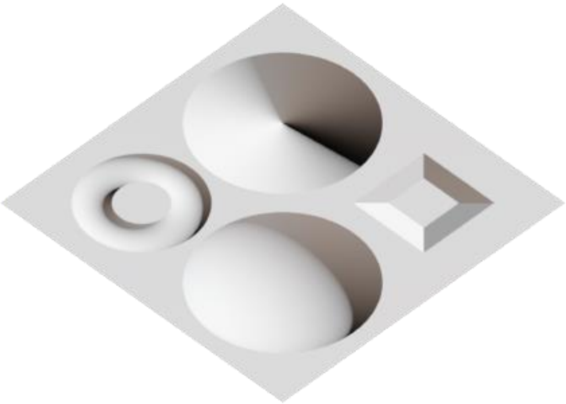
Bump Mapping



Normal Maps

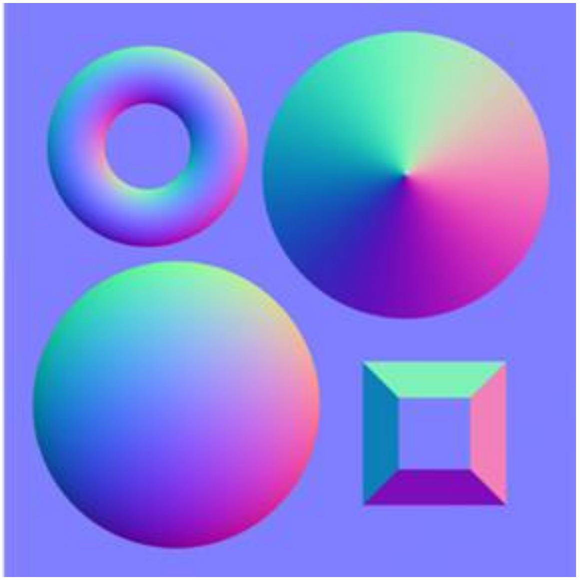






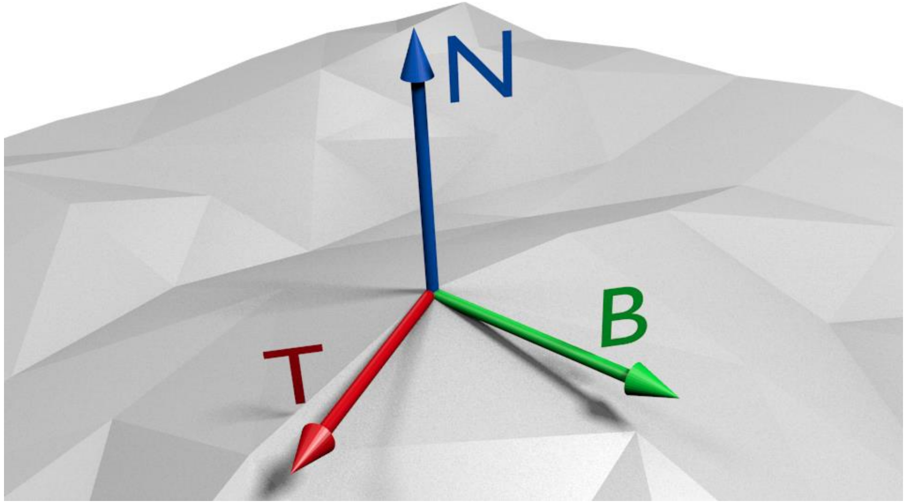
Normal Maps

- **normal = 2 * color - 1;**



Tangent Space

- **Defines coordinate systems orthogonal to the surface**

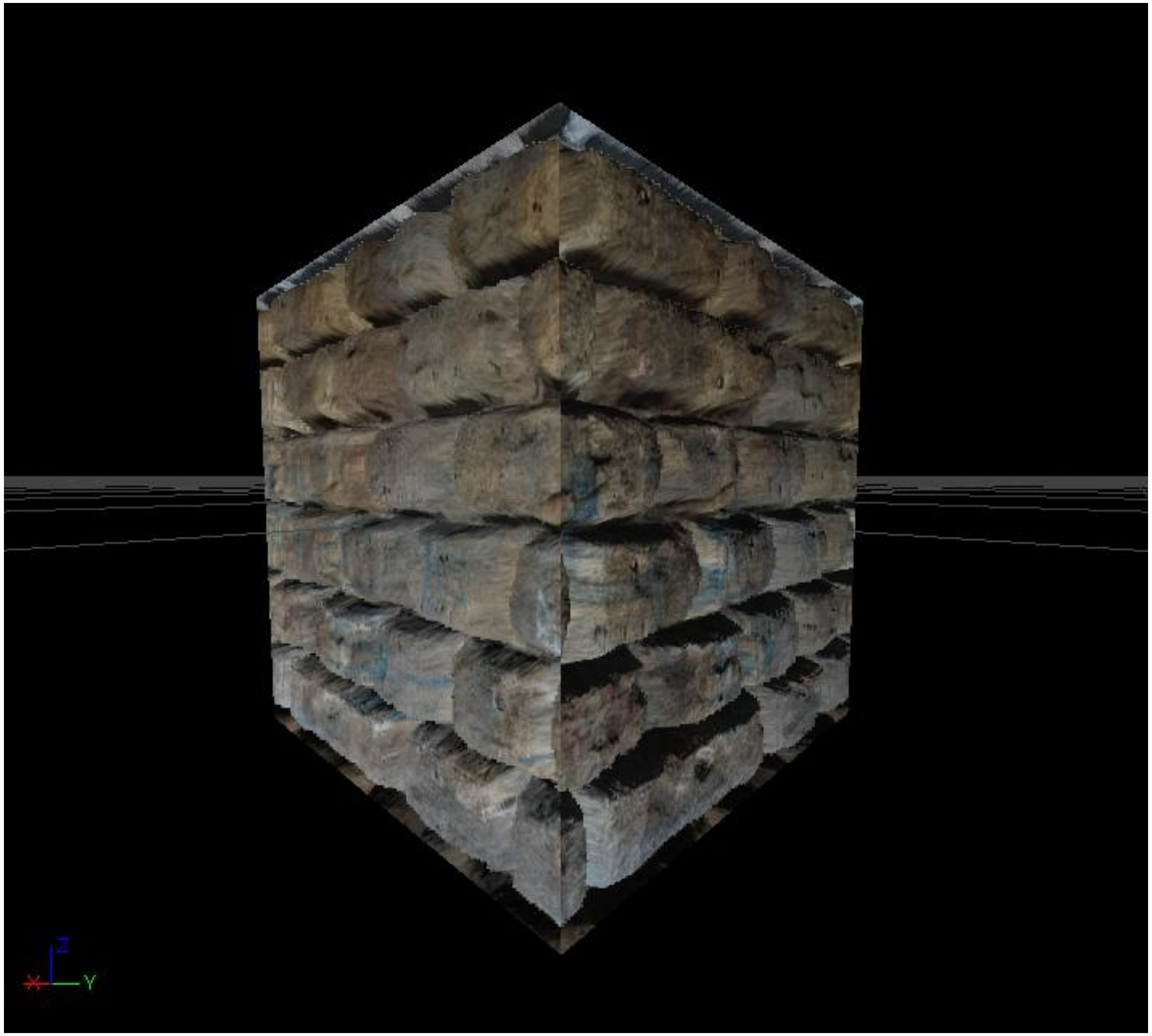


- **Reuse texture coordinates:**
  deltaPos1 = deltaU1 * T + deltaV1 * B
  deltaPos2 = deltaU2 * T + deltaV2 * B

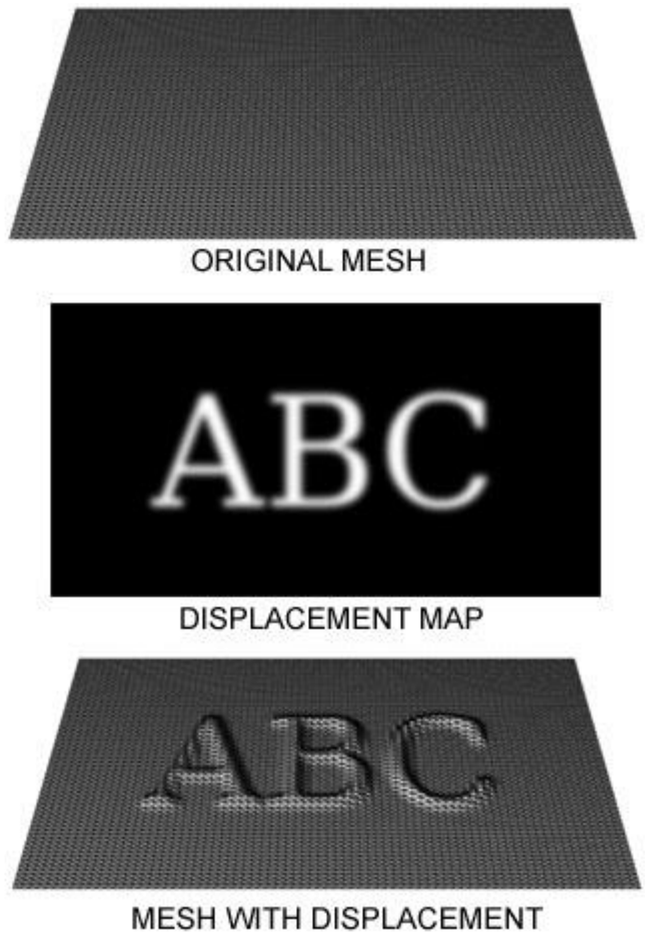
- $\begin{bmatrix} \mathbf{T} & \mathbf{B} & \mathbf{N} \\ \mathbf{T} & \mathbf{B} & \mathbf{N} \\ \mathbf{T} & \mathbf{B} & \mathbf{N} \end{bmatrix}$

Normal Mapping

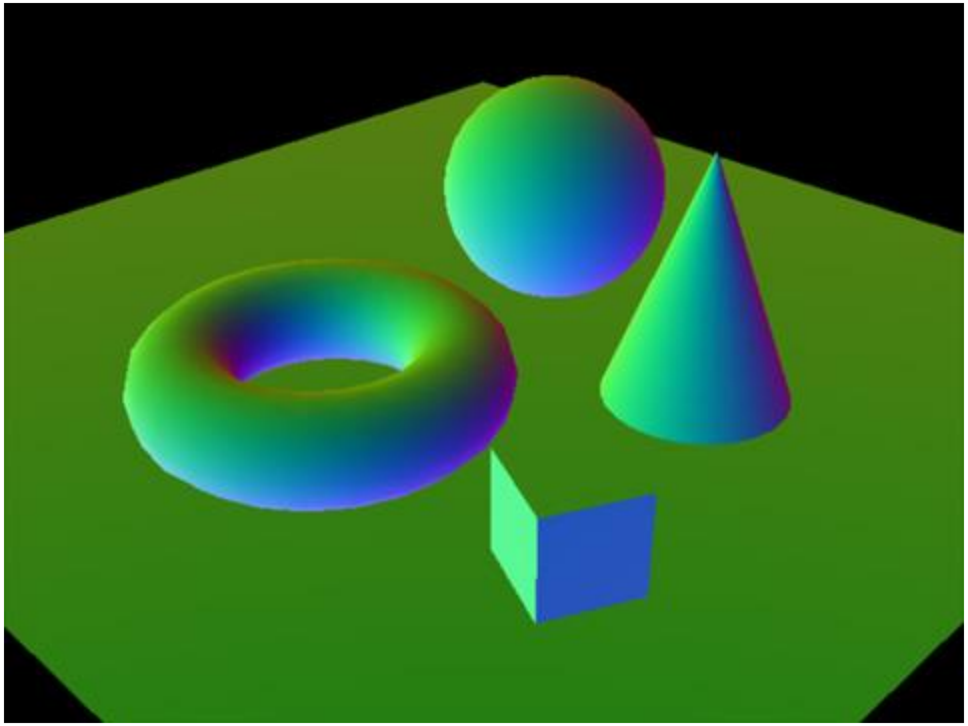
Parallax Occlusion Mapping



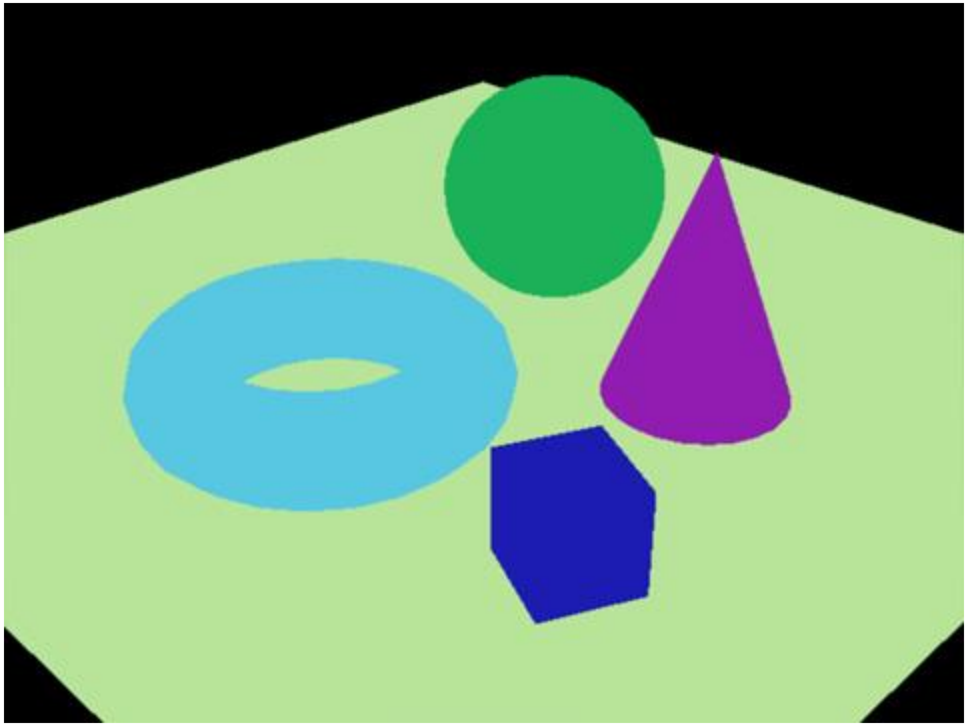
Displacement Mapping



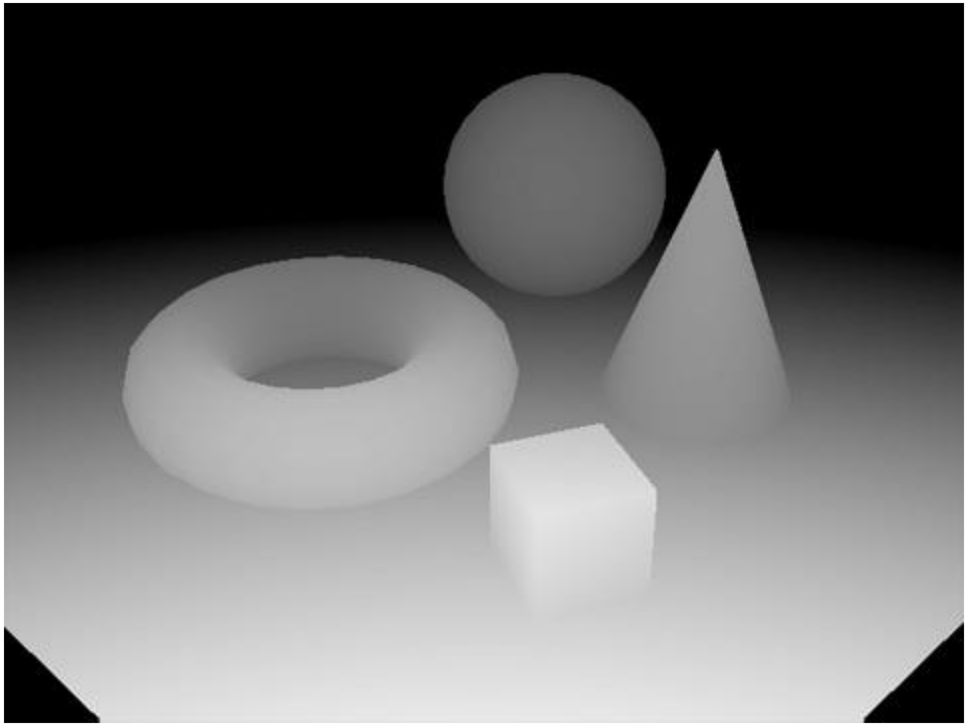
Deferred Shading



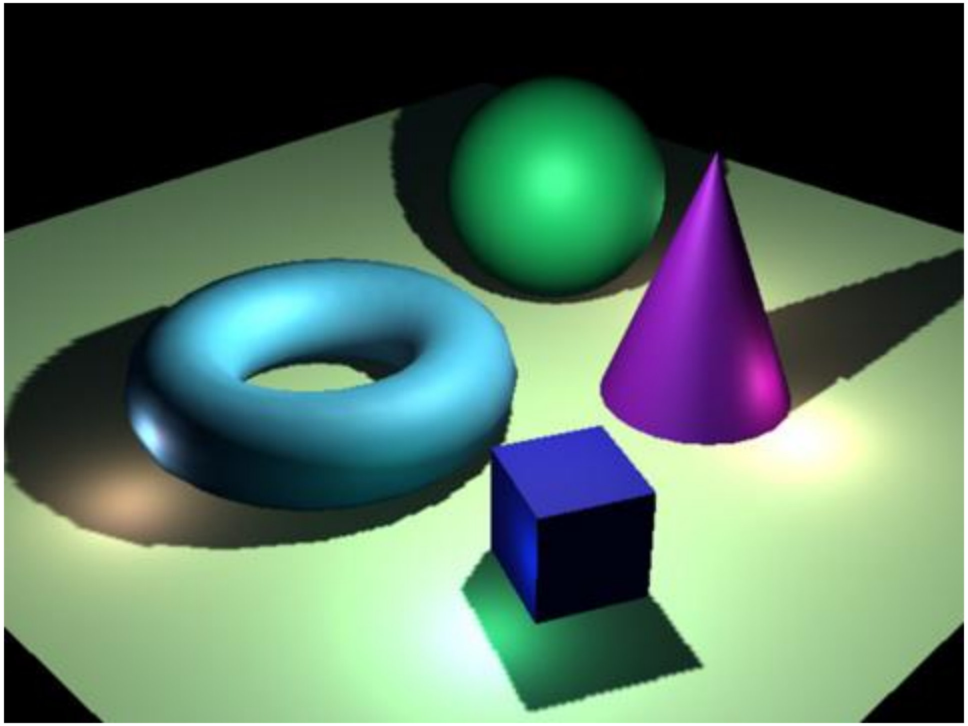
Deferred Shading



Deferred Shading



Deferred Shading



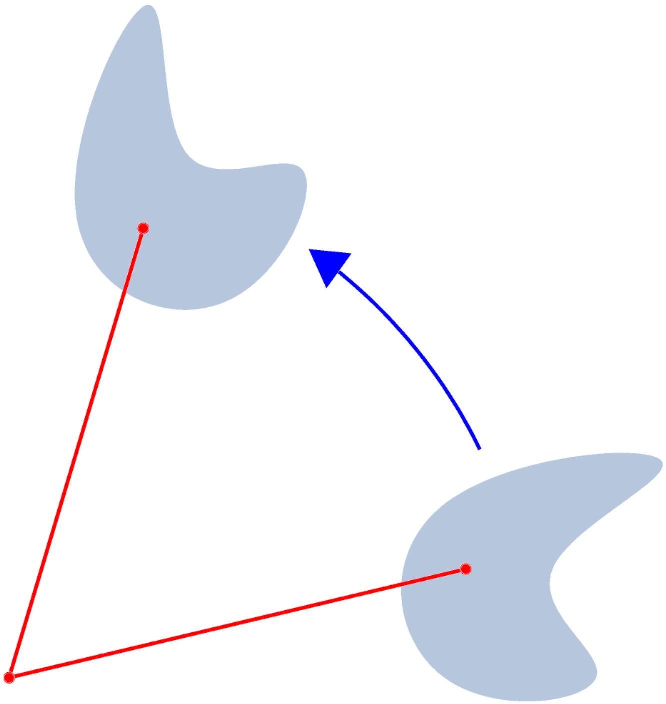
MVP

- **projection * view * model**

- **Animate the model matrix to animate an object**

Movement and Rotation

- **model = (translate to end position) * rotation**
  *** (translate rotation center to 0)**

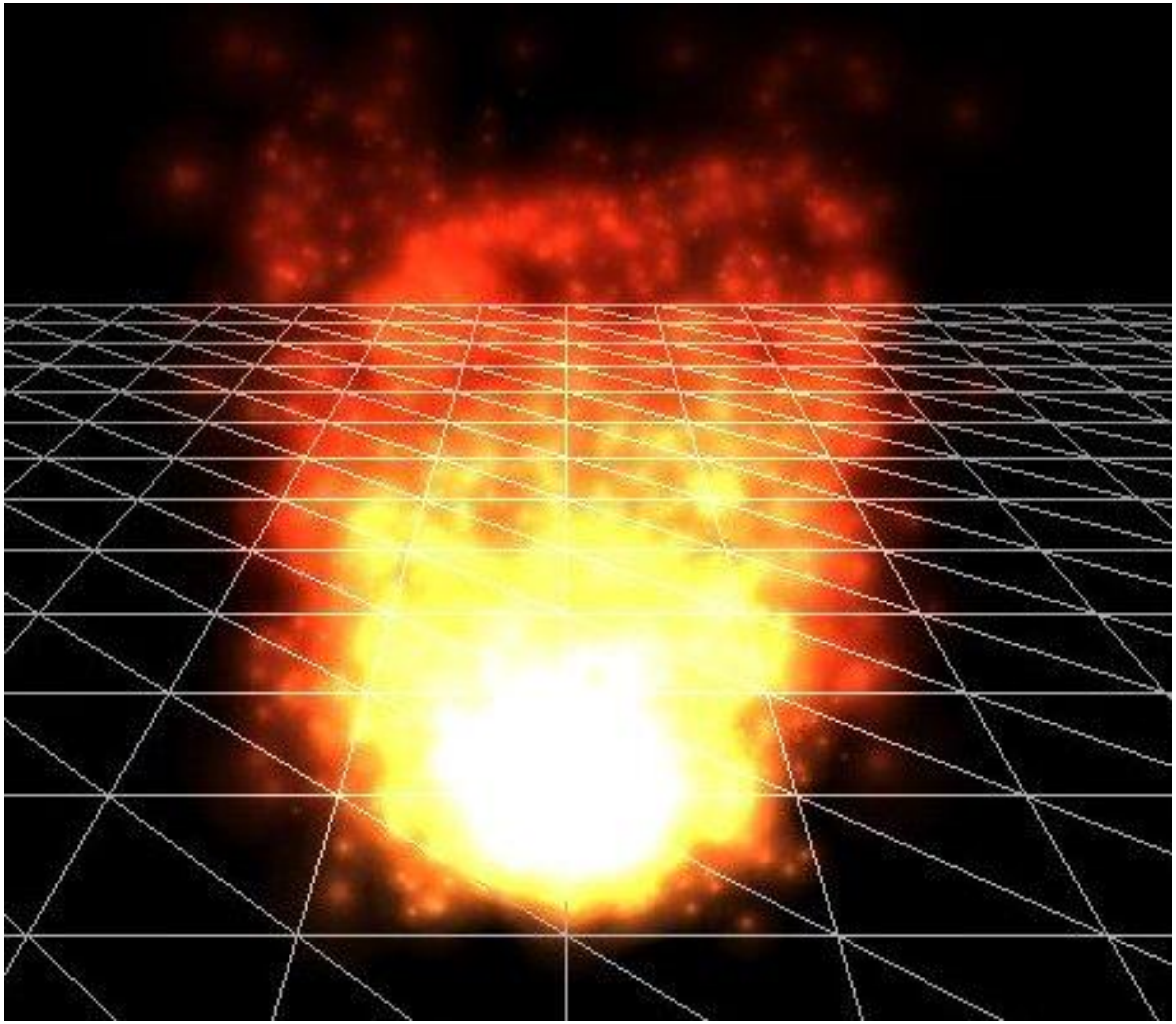


Scale

Shear

Total Time
00:06:166
Lap Time
00:06:166
175mph
Total Record
02:30:995
Course Record
00:48:706
Coin: 2
Lap: 1/3
8th/8
5
175

Particle Systems



Particle Systems

TECHNISCHE UNIVERSITÄT DARMSTADT

- **A particle is (mostly)**
  - Just an image
    - „Billboards“
    - One texture, two triangles
    - Always rotated to camera
    - Use inverse of view matrix

Particle Systems

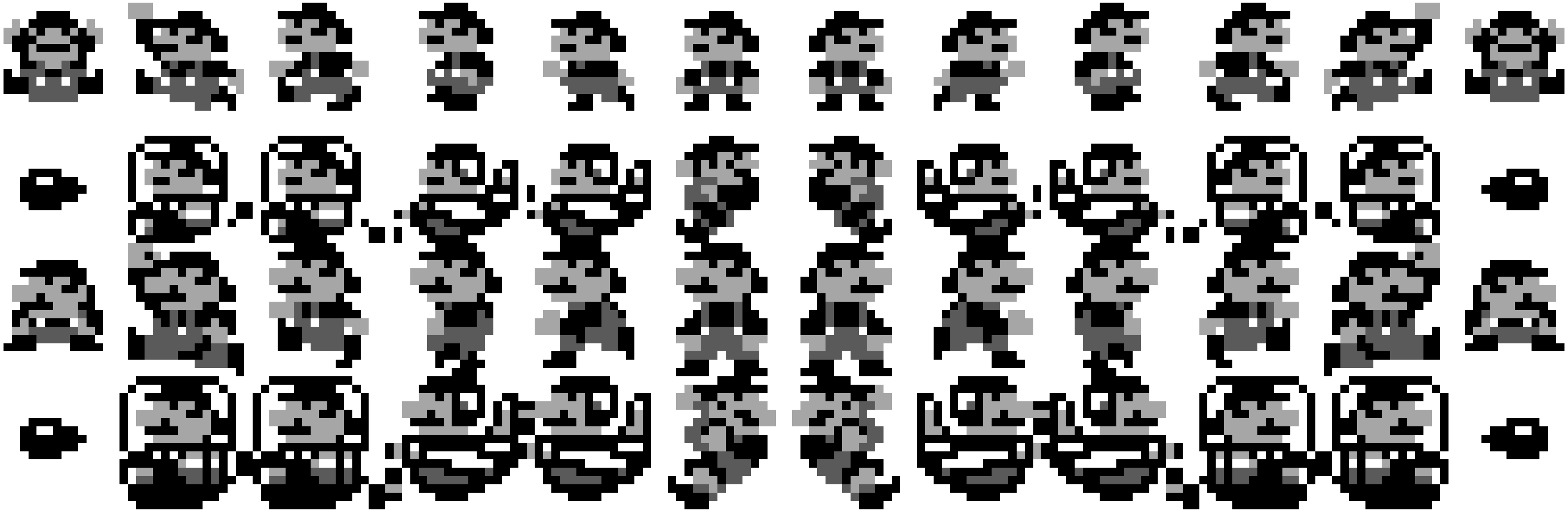
- **Emitter**
  - Source of particles
- **Animation**
  - Use super simple physics
    - Constant speed or constant speed & gravity
  - Fade out
    - alpha = 1 - time / max_time

Particle Systems

Fluids

Sprite Sheets

TECHNISCHE UNIVERSITÄT DARMSTADT



Vertex Animations

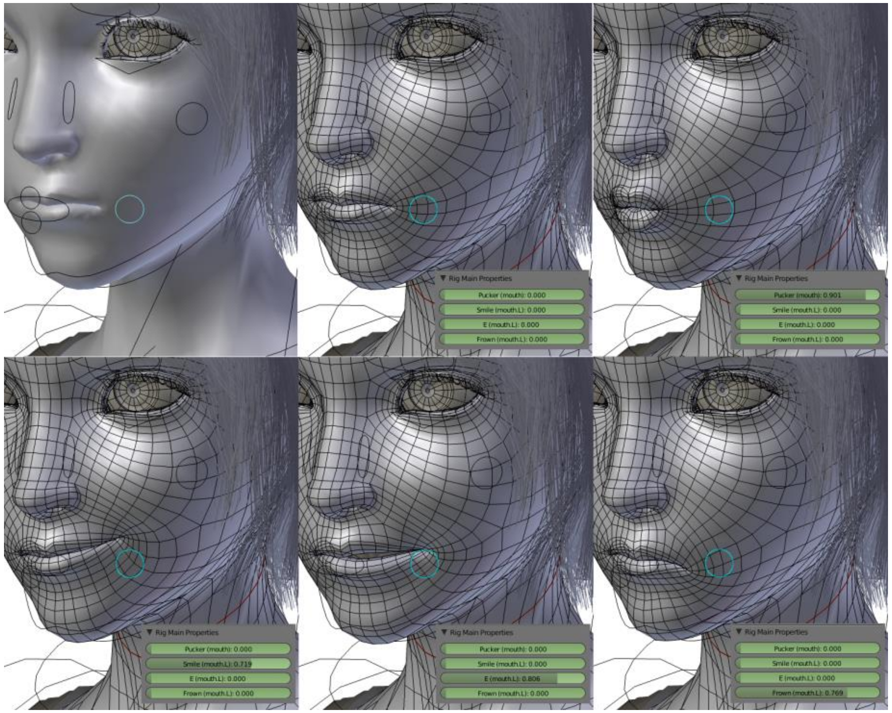
Vertex Animations

- **100 frames * 100000 vertices = lots of data**

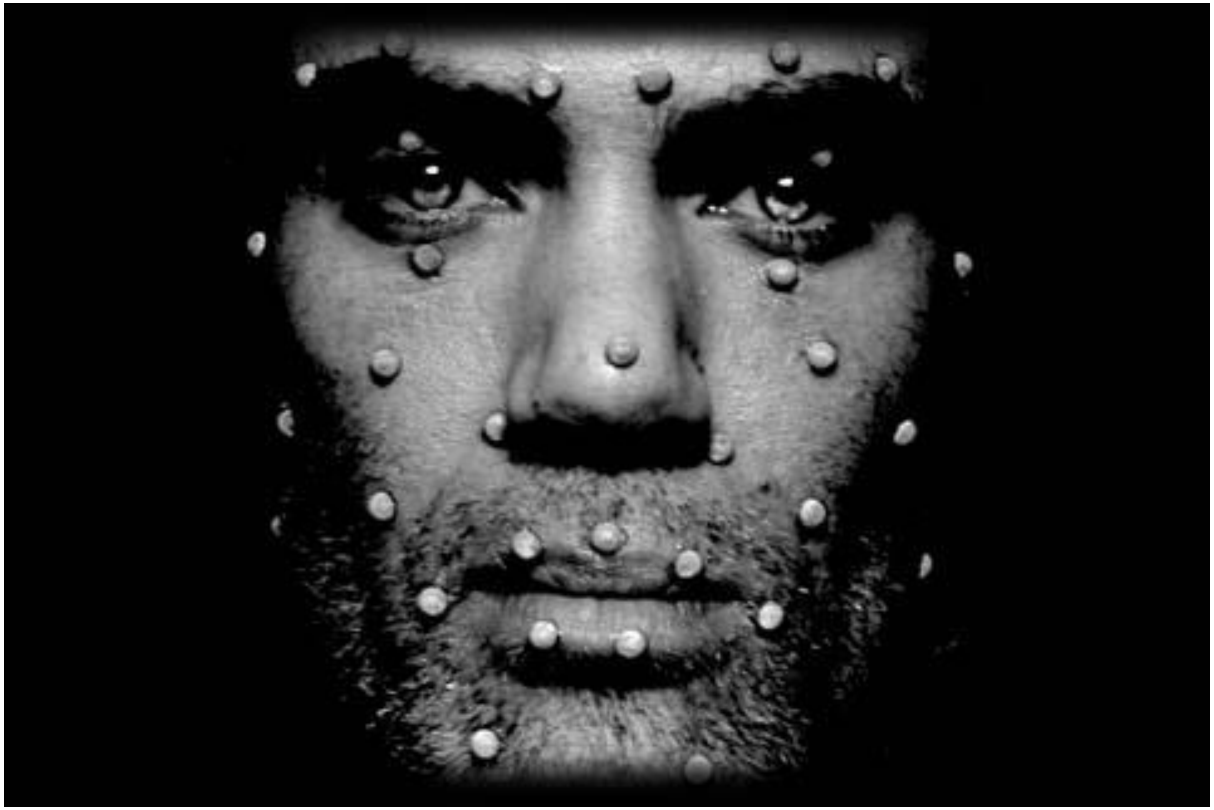
Blend Vertex Positions

Faces

- **Morph target animation**



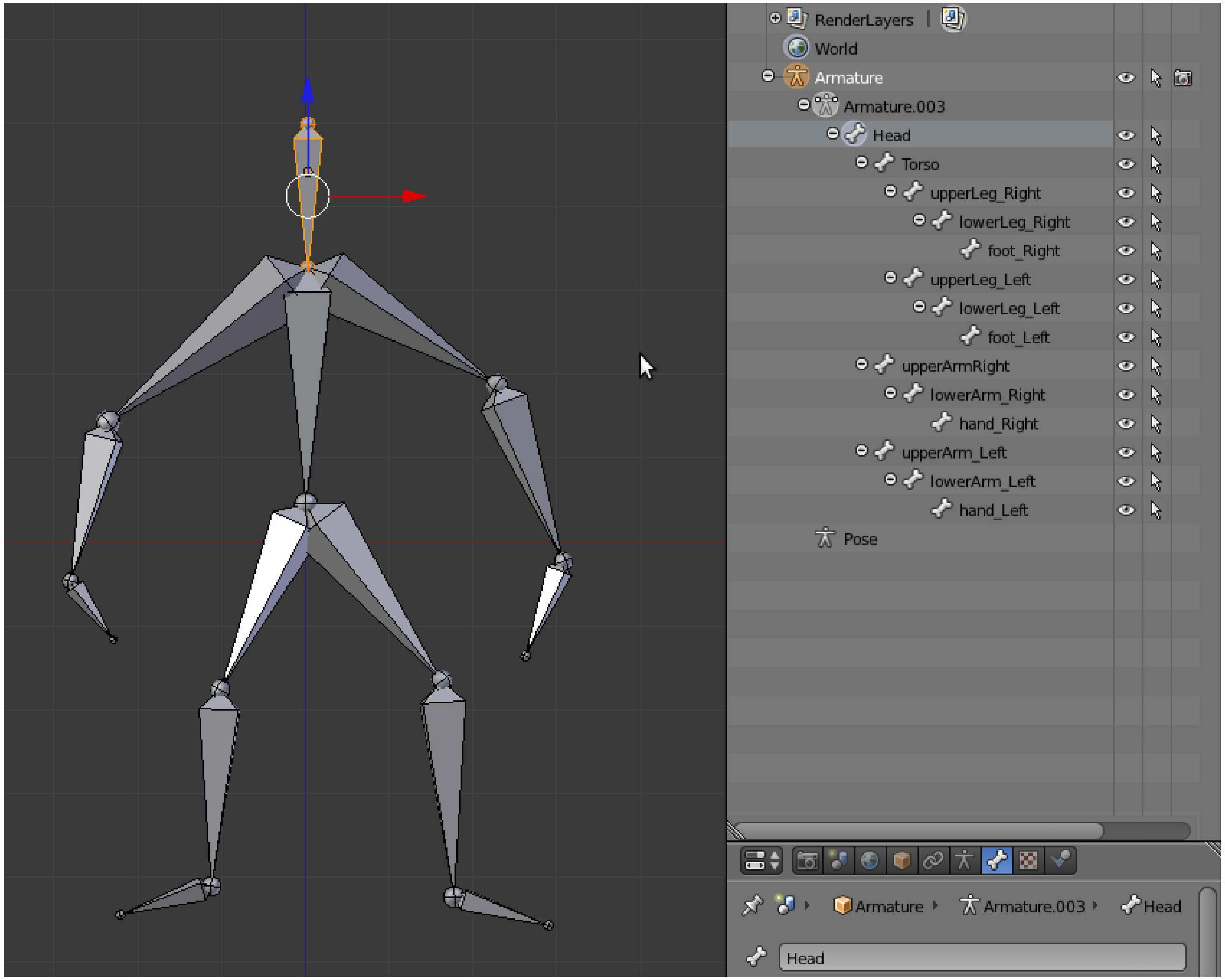
Performance Capture



Skeletal Animation

Skeletal Animation

Skeletal Animation



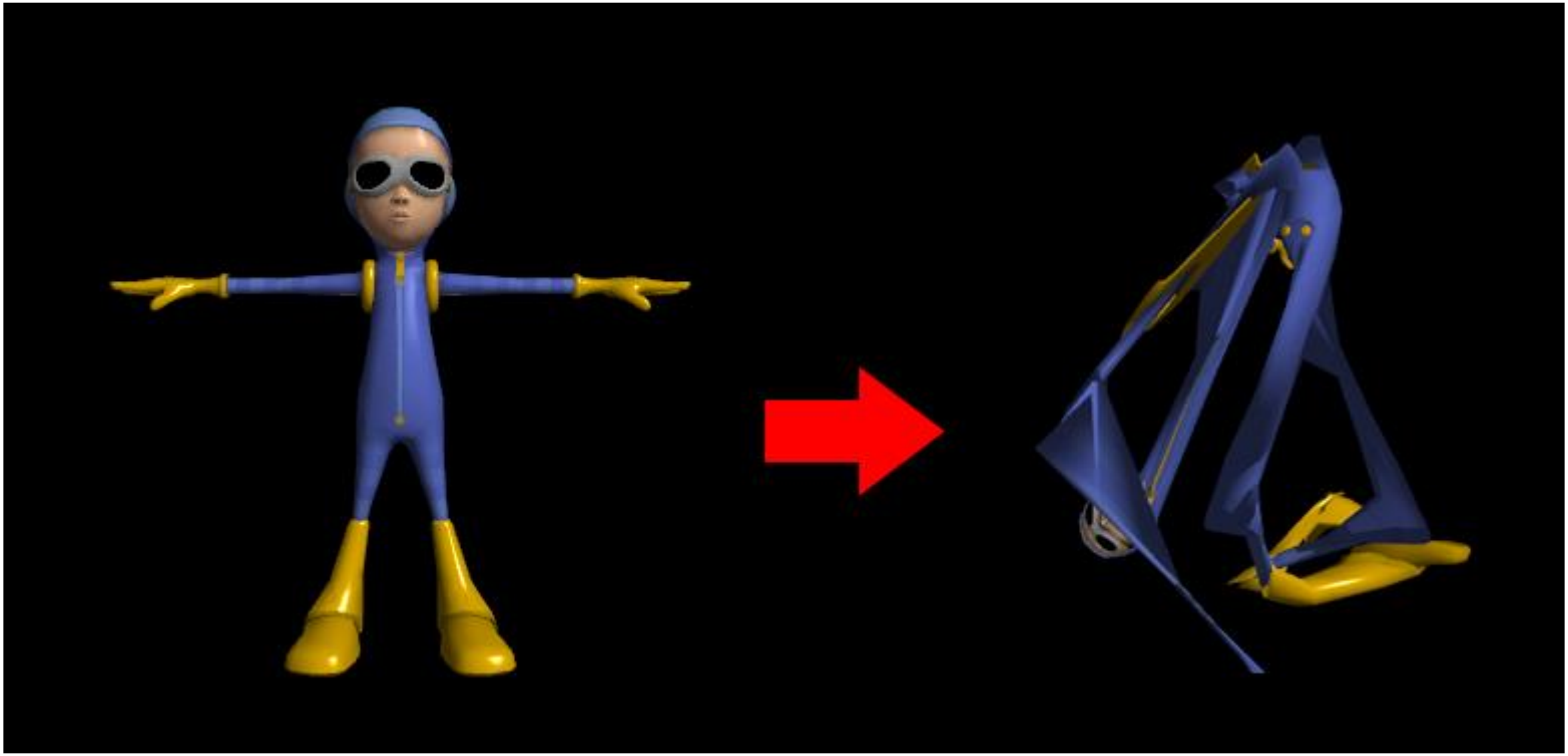
Skeletal Animation

TECHNISCHE UNIVERSITÄT DARMSTADT

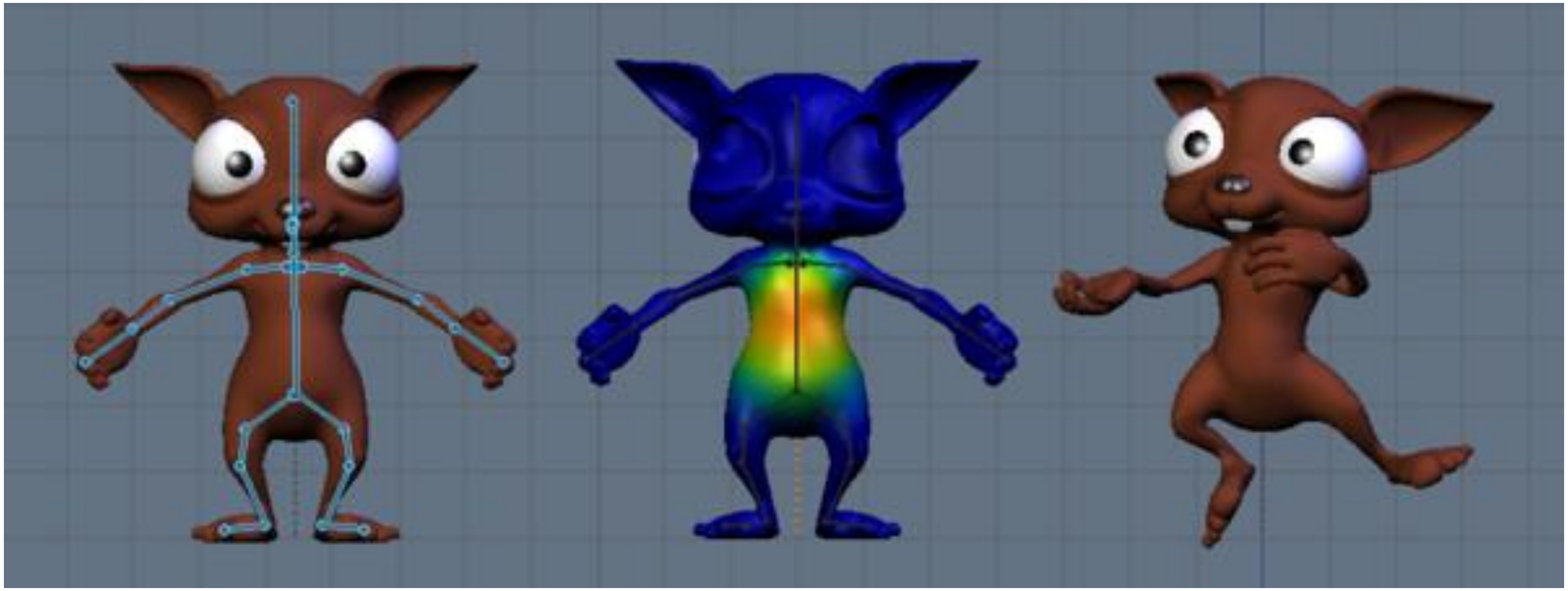
- **One bone – One Transformation matrix**
  - Or just a rotation
  - Depends on your gfx tool

- **Animation**
  - Just an array of small transformation matrix arrays
  - Framerate can be low
    - Interpolation works fine

Skinning



Skinning



Skinning

- **For each vertex**
  - Array of (weight, index)

- **At start**
  - Compute inverse of every bone transform matrix

- **For animation step**
  - Compute new transform matrices
  - For each bone compute new transform * old inverse

- **For each vertex**
  - For each weight
    - Compute (new transform * old inverse * vertex) * weight
  - Sum it up

Motion Capturing

Inverse Kinematics

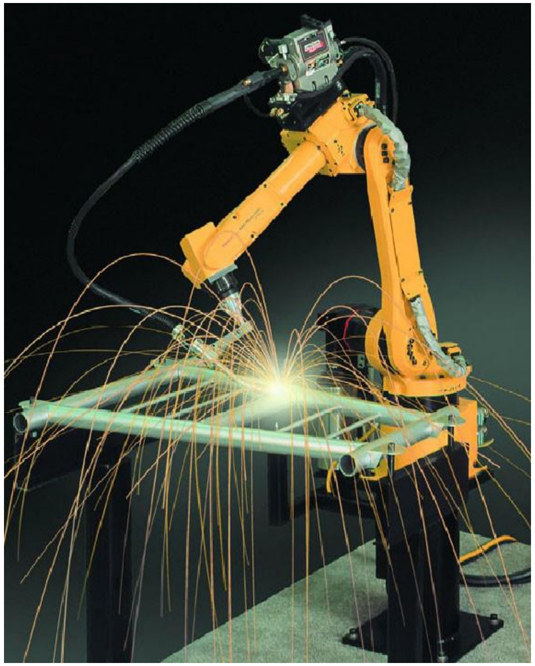
- **Forward Kinematics**
  - Input: Bone rotations
  - Output: Final positions

- **Inverse Kinematics:**
  - Input: Final positions
  - Output: Bone rotations

Inverse Kinematics

Inverse Kinematics

- **Numerical, iterative solution using Jacobi Matrix**
  - See Robotics Lecture



Unexpected Deformations

- **„Achselhölle“**

- **Spherical Skinning**
  - http://www.crytek.com/download/izfrey_siggraph2011.pdf

Muscles

Muscles

Physical Animations

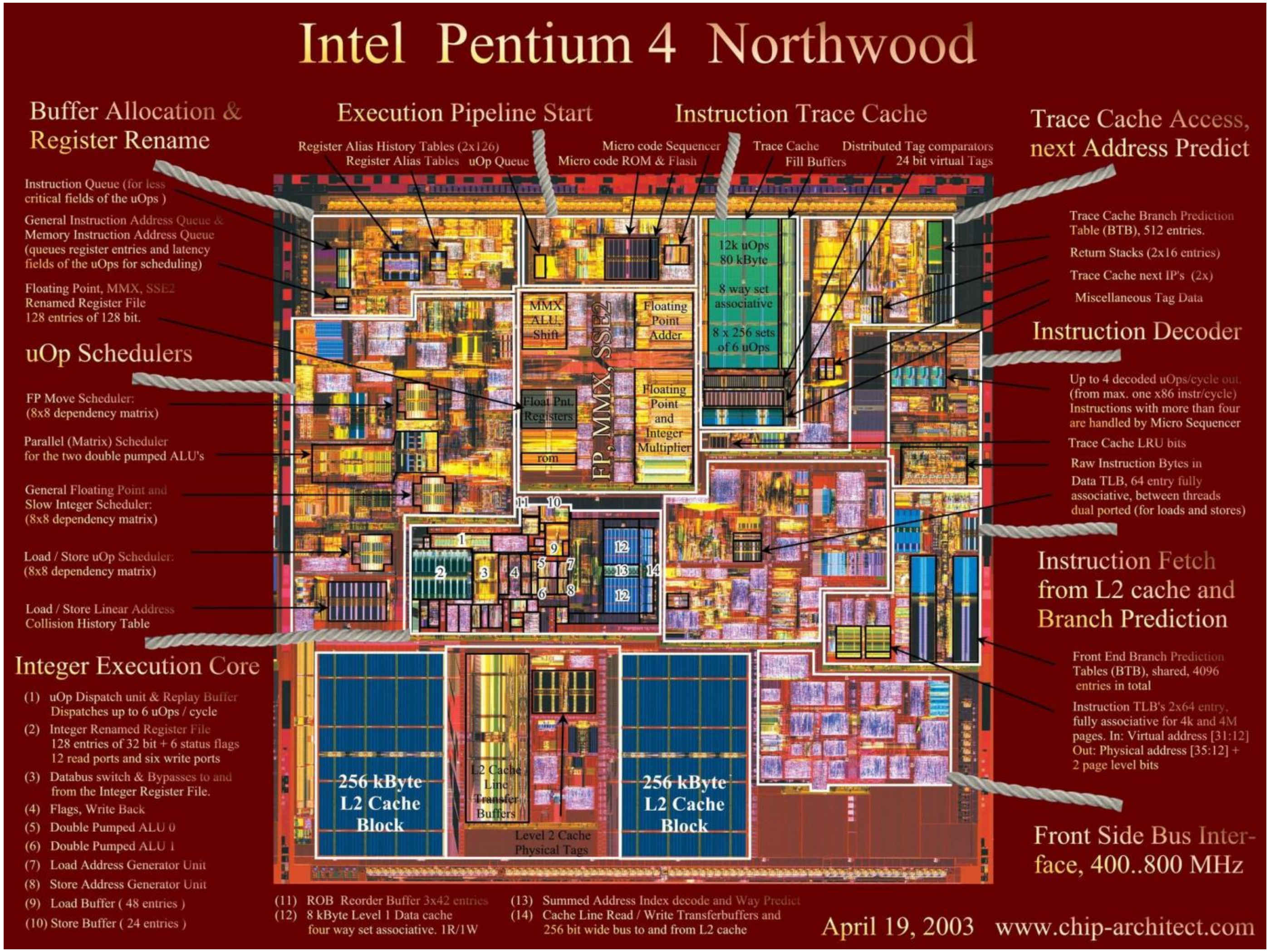
Hair, Cloth,…

Rag Dolls

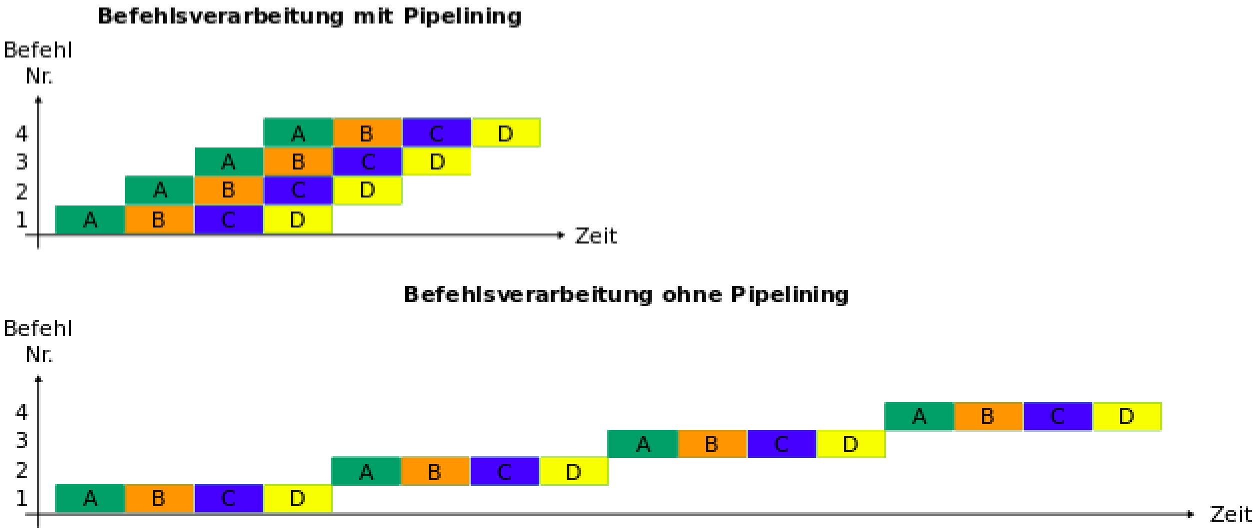
Rag Doll <-> Skeletal Animation

- **Player hit -> rag doll simulation**
- **Wait**
- **Blend from current positions to nearest known animation state**
- **Play animation**

CPU internals



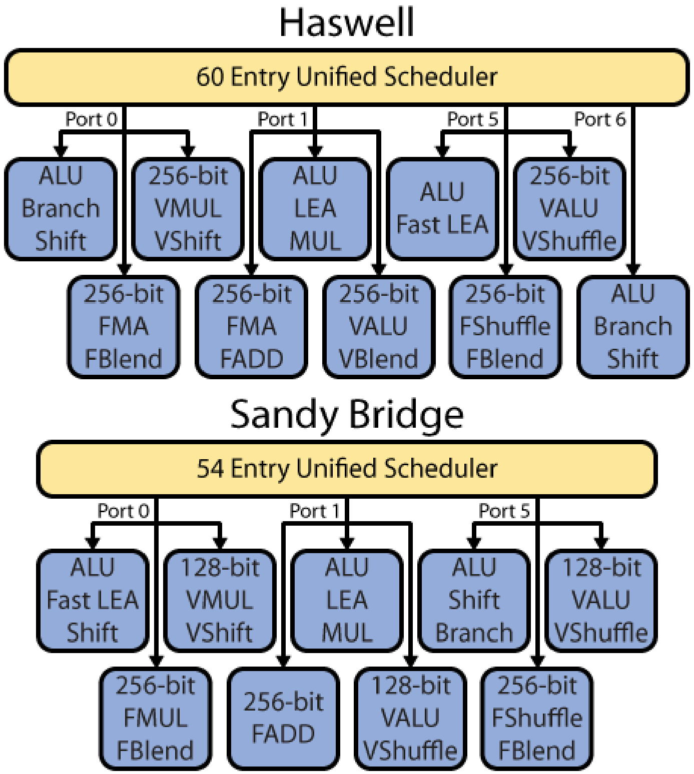
Pipelining



Befehlsverarbeitung mit Pipelining
Befehl Nr.
4
3
2
1
A
B
C
D
Zeit
Befehlsverarbeitung ohne Pipelining
Befehl Nr.
4
3
2
1
A
B
C
D
Zeit

Multiple Execution Units

- „ Note that Figure 3 does not show every execution unit, due to space limitations." (from http://www.realworldtech.com/haswell-cpu/4/)



Hazards

- **Structural Hazards**
  - Out of hardware
- **Data Hazards**
  - Data dependencies
- **Control Hazards**
  - Dynamic branching

Structural Hazards

- **Modern CPUs add more ALUs**
- **Already at a very high level**

Data Hazards

- **Sometimes just register uses, not real data dependencies**

- **subl rax, rbx**
  addl rcx, rax

- **subl rcx, rax**
  addl rbx, rax

- **-> Register renaming**
  - CPU uses more registers internally than can be directly addressed

Data Hazards

- **Compiler can help**
  - Reorder instructions
  - Depends highly on CPU

- **Out-of-Order CPUs**
  - Can reorder instructions themselves
  - Can incorporate current situation in decisions
  - All current x86 CPUs are out-of-order
  - More and more ARM CPUs are out-of-order
  - PS360 are in-order

Control Hazards

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Speculative execution**
  - Branch Prediction more and more sophisticated

```
int main()
{
    // generate data
    const unsigned arraySize = 32768;
    int data[arraySize];

    for (unsigned c = 0; c < arraySize; ++c)
        data[c] = std::rand() % 256;


    // !!! with this, the next loop runs faster
    std::sort(data, data + arraySize);


    // test
    clock_t start = clock();
    long long sum = 0;

    for (unsigned i = 0; i < 100000; ++i)
    {
        // primary loop
        for (unsigned c = 0; c < arraySize; ++c)
        {
            if (data[c] >= 128)
                sum += data[c];
        }
    }

    double elapsedTime = static_cast<double>(clock() - start) / CLOCKS_PER_SEC;

    std::cout << elapsedTime << std::endl;
    std::cout << "sum = " << sum << std::endl;
}
```

Memory Access

- Cache Hierarchy critical for performance

- L1 cache ~ KiloBytes
- L2 cache ~ MegaBytes
- Main memory ~ GigaBytes

- L1 cache ~ 0.5 ns
- L2 cache ~ 7 ns
- Main memory ~ 100 ns

Memory Access

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Access pattern prediction**
  - Works best when data is reused or for sequential data reads
- **Cache Lines**
  - Memory read in blocks
  - ~ 64 Bytes
  - Proper data alignment can help

POD

- „Plain old data“

- ```
  struct Data {
          int a;
          float b;
  };
  ```

- Predictable data structures
- No constructor calls during array allocation
- No additional data for virtual function pointers

- Data data[64];
- Linear data of 64*sizeof(Data) bytes
  ```

Memory alignment

- **Add unused data**
- **Use system specific things**
  - posix_memalign(..)
- **Use alignas in C++ 11**
- **struct alignas(16) Data {**
  int a;
  float b;
  };
- **alignas(128) char cacheline[128];**

Premultiplied Alpha

- **White Pixel, transparent pixel**
  - source alpha * new pixel + (1 - source alpha) * old pixel
  - new pixel + (1 - source alpha) * old pixel
- **(1, 1) (1, 0) -> (1, 0.5) -> 0.5 (source alpha * new pixel)**
- **(1, 1) (0, 0) -> (0.5, 0.5) -> 0.25 (source alpha * new pixel)**
- **(1, 1) (0, 0) -> (0.5, 0.5) -> 0.5 (new pixel)**

Premultiplied Alpha

- **Grey Pixel, transparent pixel**
  - source alpha * new pixel + (1 - source alpha) * old pixel
  - new pixel + (1 - source alpha) * old pixel
- **(0.5, 1) (1, 0) -> (0.75, 0.5) -> 0.375 (source alpha * new pixel)**
- **(0.5, 1) (0, 0) -> (0.25, 0.5) -> 0.125 (source alpha * new pixel)**
- **(0.5, 1) (0, 0) -> (0.25, 0.5) -> 0.25 (new pixel)**
- **(0.5, 1) (0.5, 0) -> (0.5, 0.5) -> 0.25 (source alpha * new pixel)**

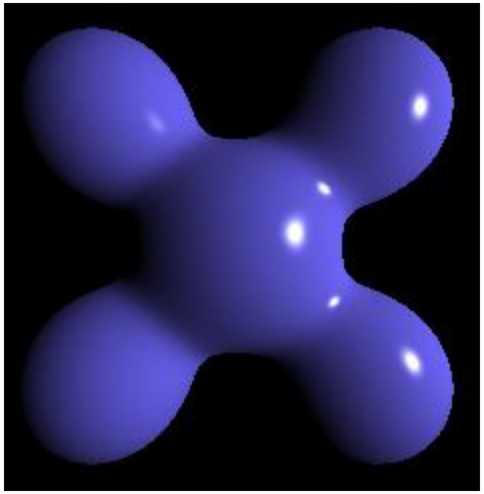
Blending Order

- source alpha * new pixel + (1 - source alpha) * old pixel
- $a * n + (1 - a) * o$
- $a2 * n2 + (1 - a2) * (a1 * n1 + (1 - a1) * o)$
- $a2*n2 + a1*n1 + o - a1*o - a1*a2*n1 - a2*o + a1*a2*o$

- source alpha * new pixel + old pixel
- $a * n + o$
- $a2 * n2 + (a1 * n1 + o)$
- $= a1 * n1 + a2 * n2 + o$

Phong Lighting

- **Reflects only light sources**
- **Expects light sources to be round**



Roughness

- Use mip mapped cube map
- Choose mip map level based on roughness
- Higher mip level -> blurrier -> increased roughness