



Introduction to Programmable GPUs

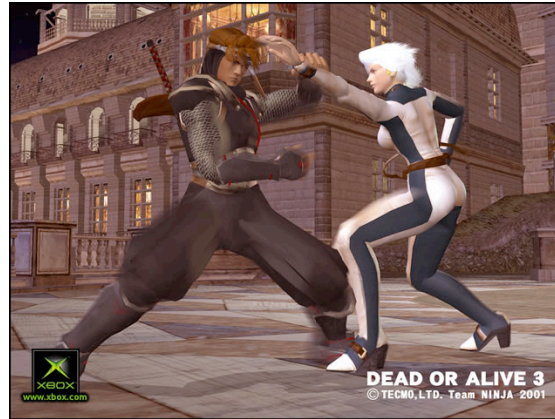
CPSC 314

- Homework: no new homework this week (focus on quiz prep)
- Quiz 2
 - this Friday
 - topics:
 - everything after transformations up until last Friday's lecture
 - questions on rendering pipeline as a whole
- Office hours (Wolfgang) Thursday, Friday 11:30-12:30

Real Time Graphics



Virtua Fighter 1995
(SEGA Corporation)
NV1



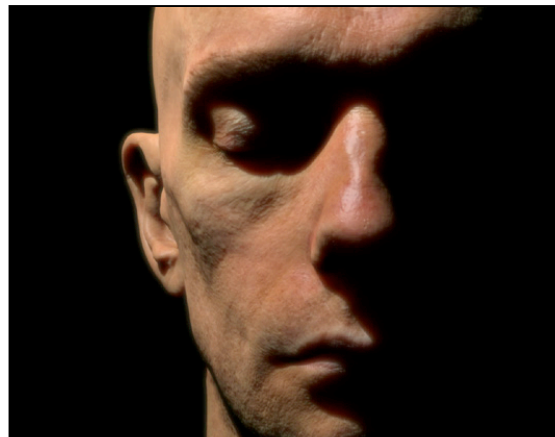
Dead or Alive 3 2001
(Tecmo Corporation)
Xbox (NV2A)



Dawn 2003
(NVIDIA Corporation)
GeForce FX (NV30)



Nalu 2004
(NVIDIA Corporation)
GeForce 6



Human Head 2006
(NVIDIA Corporation)
GeForce 7

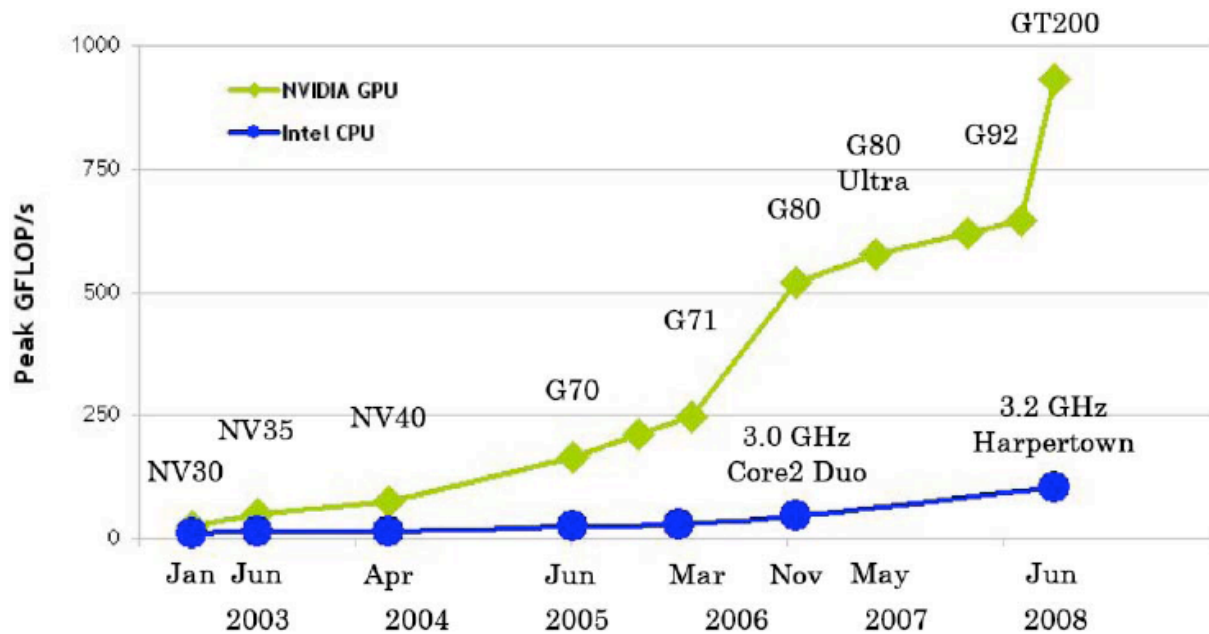


Medusa 2008
(NVIDIA Corporation)
GeForce GTX 200

GPUs vs CPUs



- 800 GFLOPS vs 80 GFLOPS
- 86.4 GB/s vs 8.4 GB/s



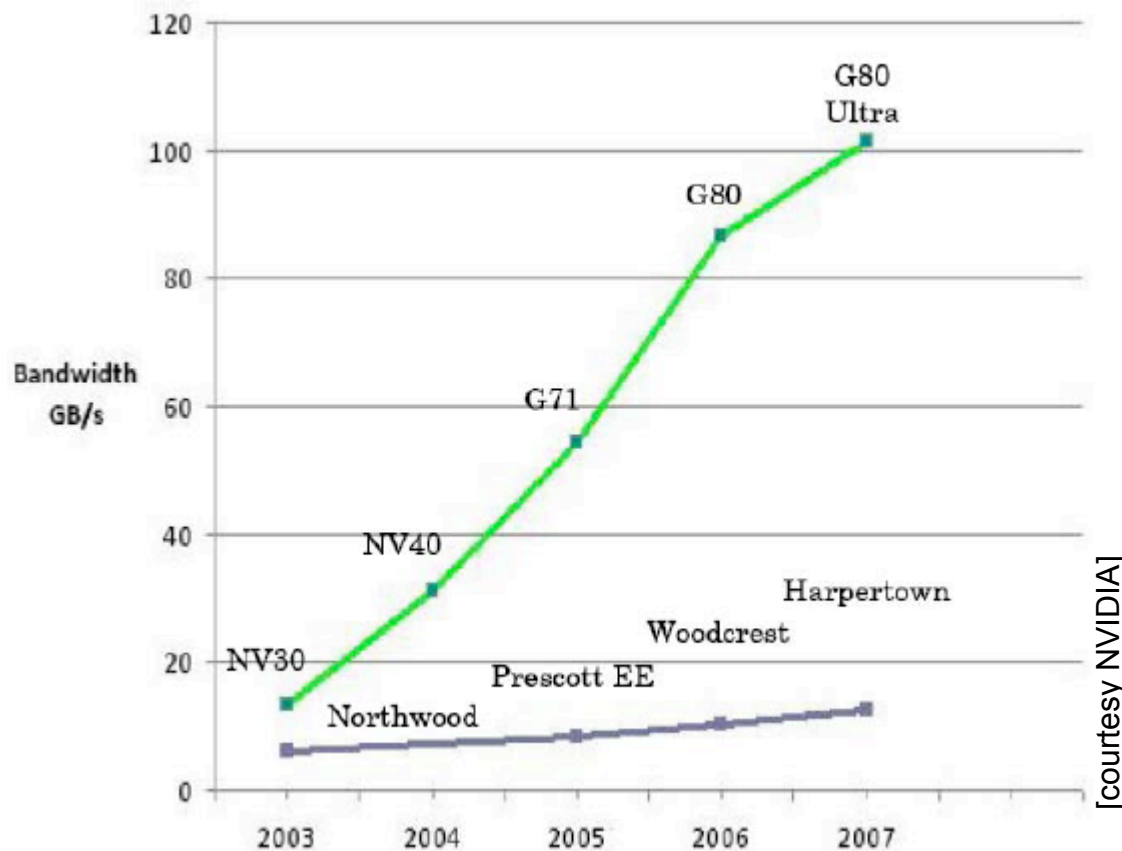
GT200 = GeForce GTX 280	G71 = GeForce 7900 GTX	NV35 = GeForce FX 5950 Ultra
G92 = GeForce 9800 GTX	G70 = GeForce 7800 GTX	NV30 = GeForce FX 5800
G80 = GeForce 8800 GTX	NV40 = GeForce 6800 Ultra	

[courtesy NVIDIA]

GPUs vs CPUs



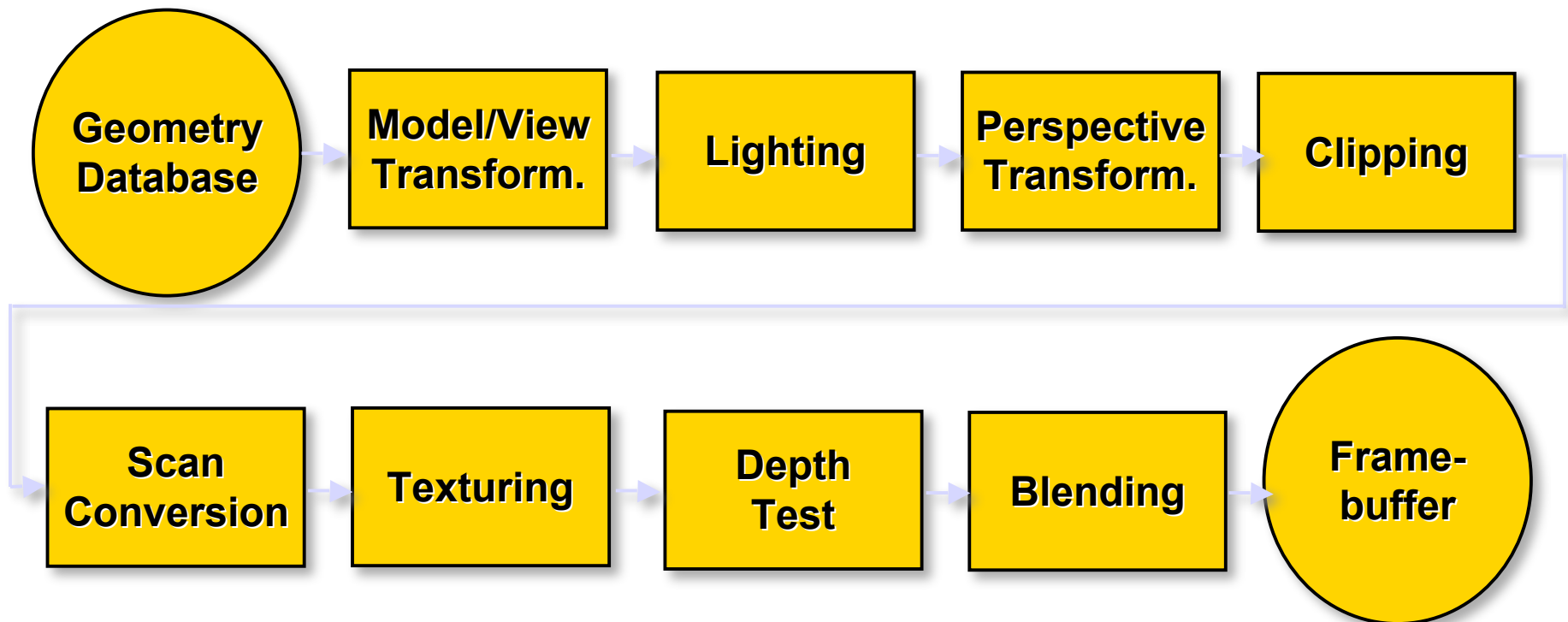
- 800 GFLOPS vs 80 GFLOPS
- 86.4 GB/s vs 8.4 GB/s



Programmable Pipeline



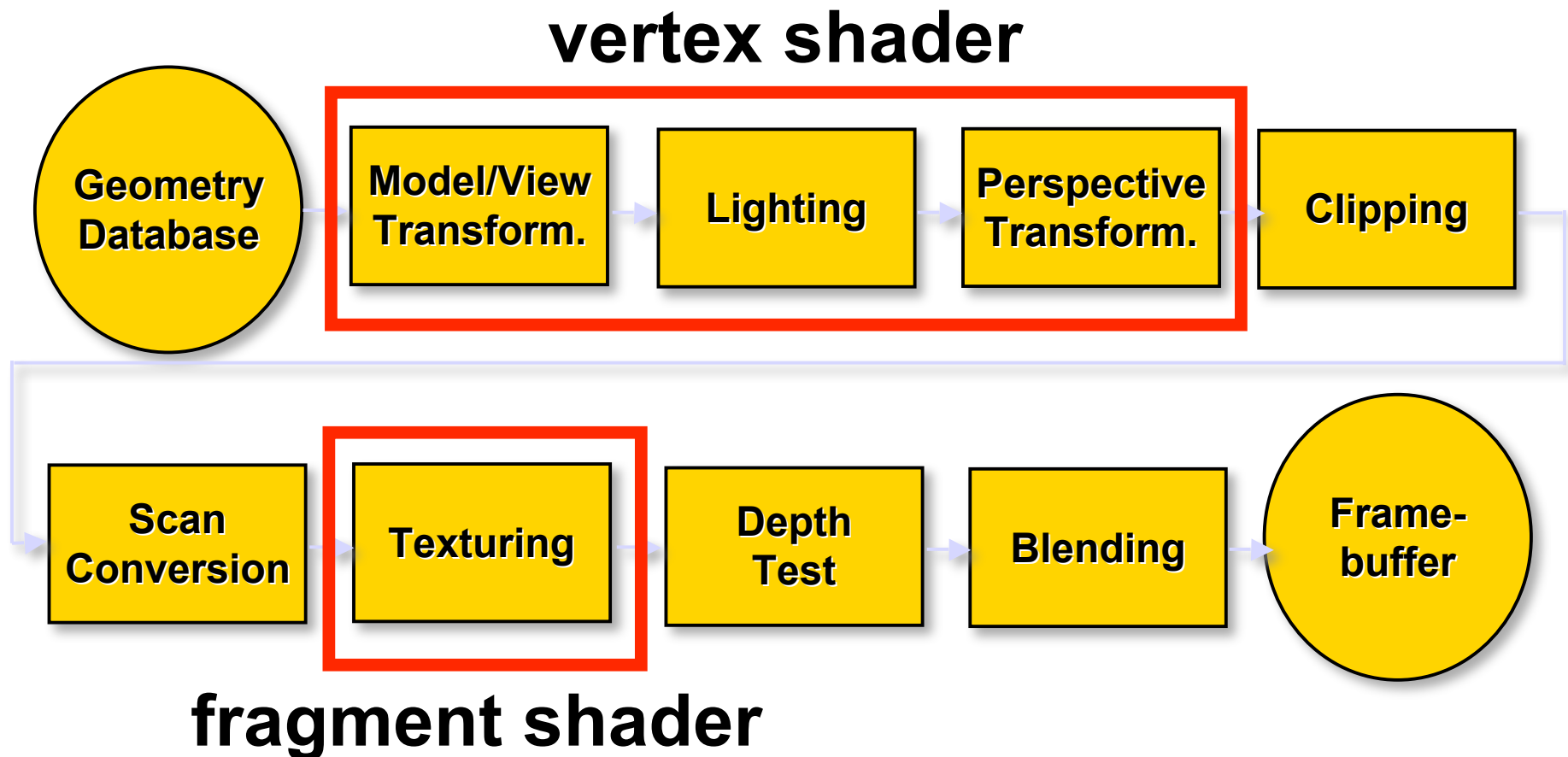
- so far:
 - have discussed rendering pipeline as specific set of stages with **fixed functionality**



Programmable Pipeline



- now: programmable rendering pipeline!



- performs all **per-vertex** computation (transform & lighting):
 - model and view transform
 - perspective transform
 - texture coordinate transform
 - per-vertex lighting

Vertex Shader



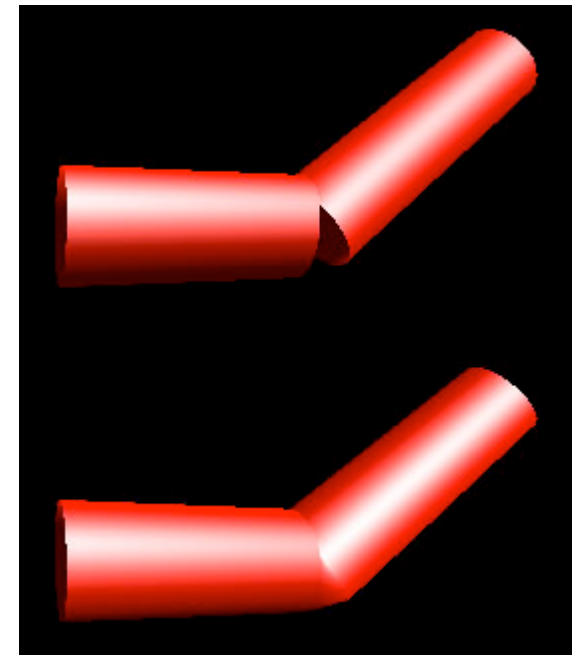
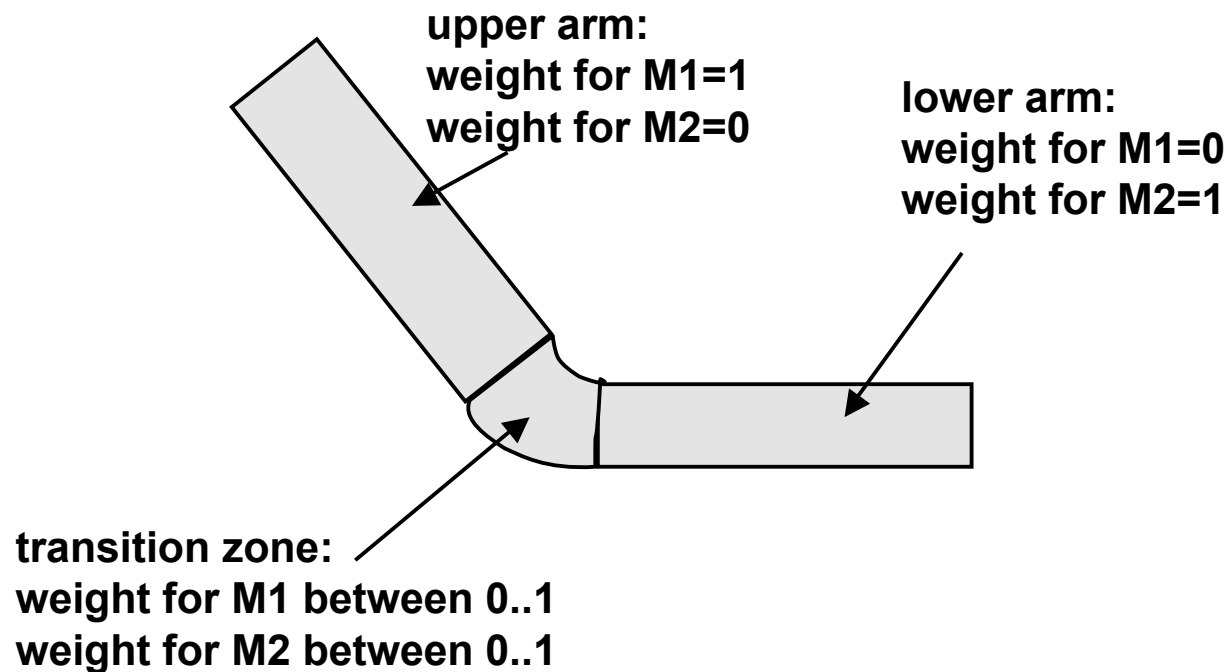
- input:
 - vertex position and normal (sometimes tangent)
 - (multi-)texture coordinate(s)
 - modelview, projection, and texture matrix
 - vertex material or color
 - light sources – color, position, direction etc.

- output:
 - 2D vertex position
 - transformed texture coordinates
 - vertex color

Vertex Shader - Applications



- deformable surfaces: skinning
- different parts have different rigid transformations
- vertex positions are blended
- used in facial animations – many transformations!



[courtesy NVIDIA]

Fragment Shader

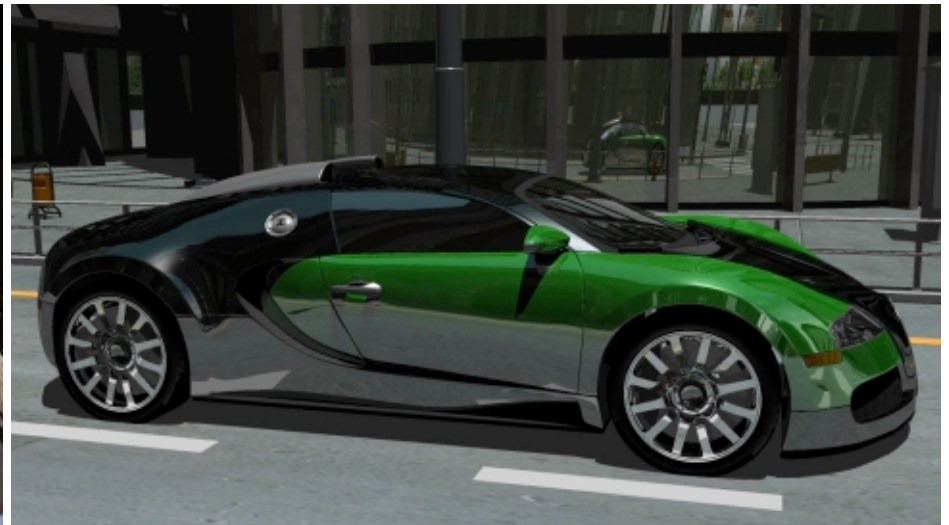


- performs all **per-fragment** computation:
 - texture mapping
 - fog
- input (interpolated over primitives by rasterizer):
 - texture coordinates
 - color
- output:
 - fragment color

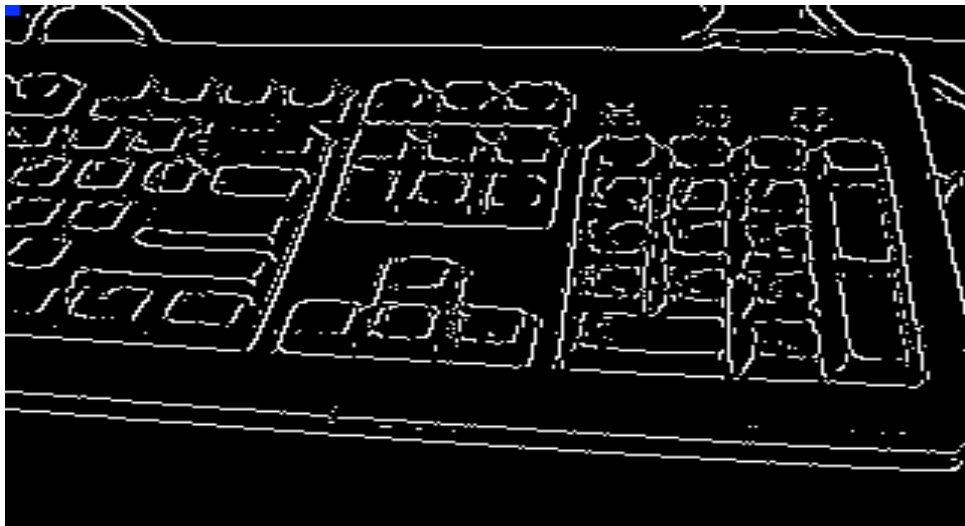
Fragment Shader - Applications



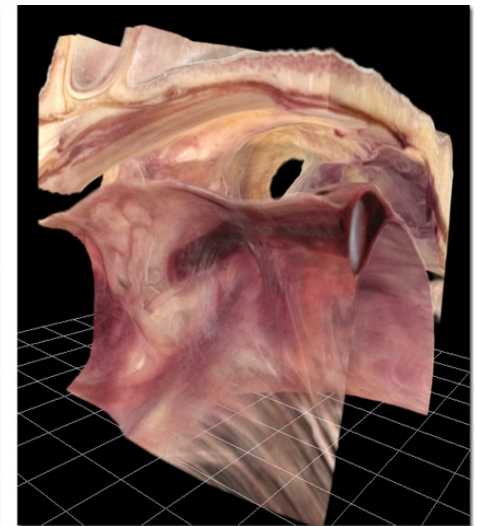
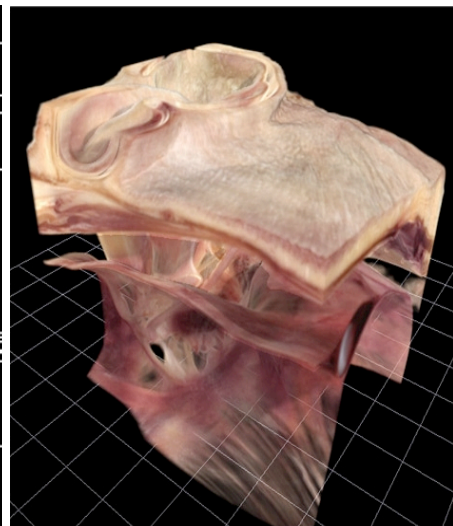
Not really shaders, but very similar to NPR!
A Scanner Darkly, Warner Independent Pictures



GPU raytracing, NVIDIA



OpenVIDIA Image Processing



Volume Ray Casting, Peter Trier

Vertex & Fragment Shader



- massively parallel computing by parallelization
- same shader is applied to all data (vertices or fragments) – SIMD (single instruction multiple data)
- parallel programming issues:
 - main advantage: high performance
 - main disadvantage: no access to neighboring vertices/fragments

Vertex Shader - Instructions



- Arithmetic Operations on 4-vectors:
 - ADD, MUL, MAD, MIN, MAX, DP3, DP4
- Operations on Scalars
 - RCP ($1/x$), RSQ ($1/\sqrt{x}$), EXP, LOG
- Specialty Instructions
 - DST (distance: computes length of vector)
 - LIT (quadratic falloff term for lighting)
- Later generation:
 - Loops and conditional jumps

Vertex Shader - Example



- morph between cube and sphere & lighting
- vertex attributes: $v[0..N]$, matrices $c[1..N]$, registers R

#blend normal and position

$\# v = \alpha v_1 + (1-\alpha)v_2 = \alpha(v_1 - v_2) + v_2$

MOV R3, v[3] ;

MOV R5, v[2] ;

ADD R8, v[1], -R3 ;

ADD R6, v[0], -R5 ;

MAD R8, v[15].x, R8, R3

MAD R6, v[15].x, R6, R5 ;

transform normal to eye space

DP3 R9.x, R8, c[12] ;

DP3 R9.y, R8, c[13] ;

DP3 R9.z, R8, c[14] ;

transform position and output

DP4 o[HPOS].x, R6, c[4] ;

DP4 o[HPOS].y, R6, c[5] ;

DP4 o[HPOS].z, R6, c[6] ;

DP4 o[HPOS].w, R6, c[7] ;

normalize normal

DP3 R9.w, R9, R9 ;

RSQ R9.w, R9.w ;

MUL R9, R9.w, R9 ;

apply lighting and output color

DP3 R0.x, R9, c[20] ;

DP3 R0.y, R9, c[22] ;

MOV R0.zw, c[21] ;

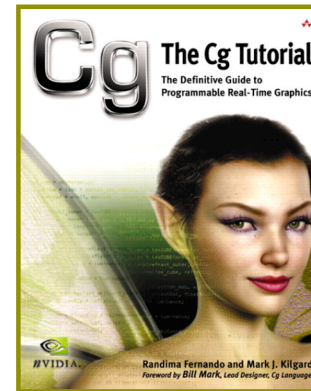
LIT R1, R0 ;

DP3 o[COL0], c[21], R1 ;

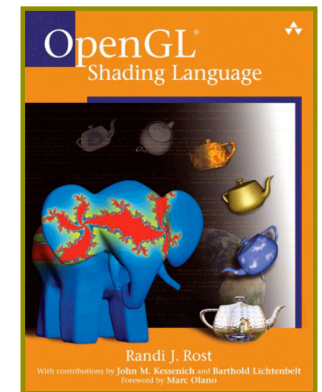
Shading languages



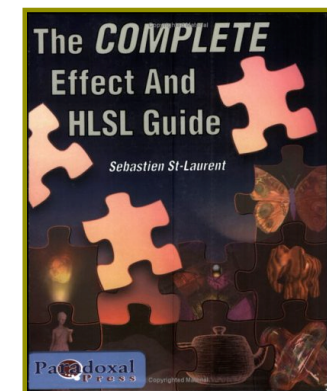
- Cg (C for Graphics – NVIDIA)



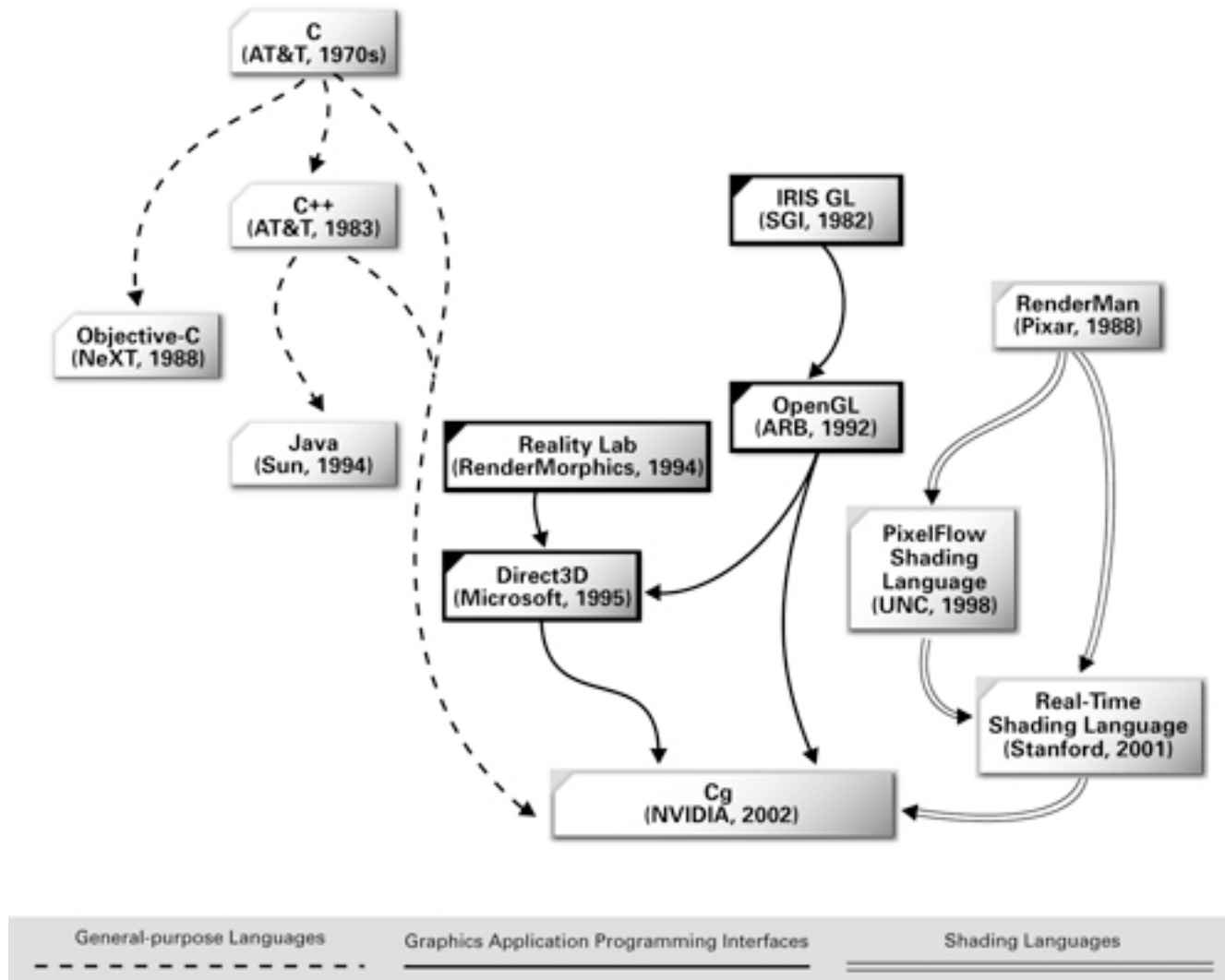
- GLSL (GL Shading Language – OpenGL)



- HLSL (High Level Shading Language – MS Direct3D)

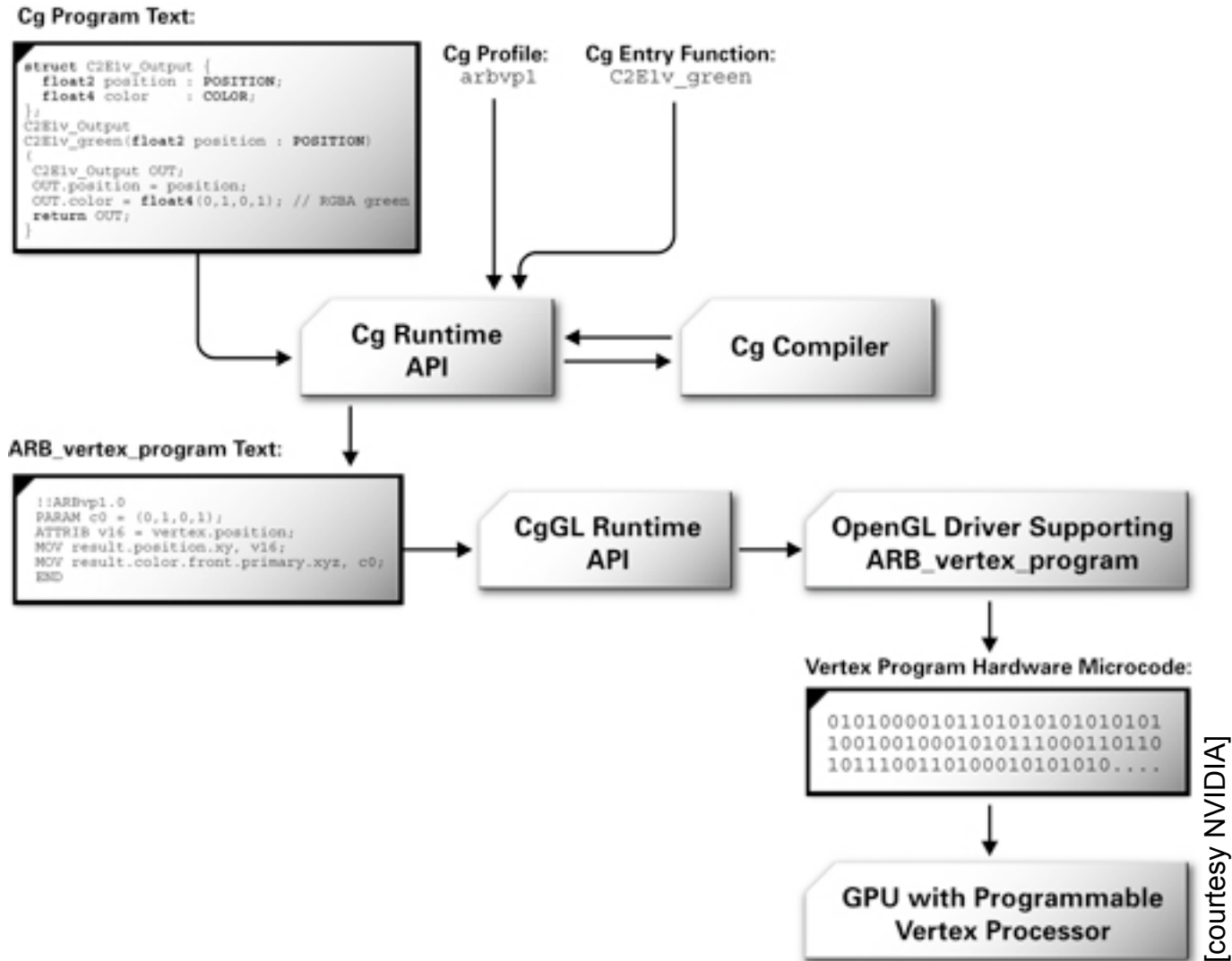


Cg History



[courtesy NVIDIA]

Cg – How does it work?



Cg – Integration into OpenGL



```
void initShader(void) {  
  
    // get fragment shader profile  
    _cgFragmentProfile = \  
        cgGLGetLatestProfile(CG_GL_FRAGMENT);  
  
    // init Cg context  
    _cgContext = cgCreateContext();  
  
    // load shader from file  
    _cgProgram = \  
        cgCreateProgramFromFile( _cgContext,  
                                CG_SOURCE,  
                                "MyShader.cg",  
                                _cgFragmentProfile,  
                                NULL, NULL);  
  
    // upload shader on GPU  
    cgGLLoadProgram( _cgProgram );  
  
    // get handles to shader parameters  
    _cgTexture = \  
        cgGetNamedParameter(_cgProgram, "texture");  
    _cgParameter = \  
        cgGetNamedParameter(_cgProgram, "parameter");  
}
```

```
void displayLoop(void) {  
  
    // setup transformation  
    ...  
  
    // enable shader and set parameters  
    cgGLEnableProfile( _cgFragmentProfile );  
    cgGLBindProgram( _cgProgram );  
  
    // set Cg texture  
    cgGLSetTextureParameter(_cgTexture, _textureID);  
    cgGLEnableTextureParameter(_cgTexture);  
  
    // set gamma  
    cgGLSetParameter1f(_cgParameter, _parameter);  
  
    // draw geometry  
    ...  
  
    // disable Cg texture and profile  
    cgGLDisableTextureParameter(_cgTexture);  
    cgGLDisableProfile( _cgFragmentProfile );  
  
    // swap buffers  
    ...  
}
```

Cg Example – Fragment Shader



- Fragment Shader: gamma mapping



```
void main( float4           texcoord : TEXCOORD,
           uniform samplerRECT texture,
           uniform float    gamma,
           out float4       color    : COLOR )
{
    // perform texture look up
    float3 textureColor = f4texRECT( texture, texcoord.xy ).rgb;

    // set output color
    color.rgb = pow( textureColor, gamma );
}
```

Cg Example – Vertex Shader

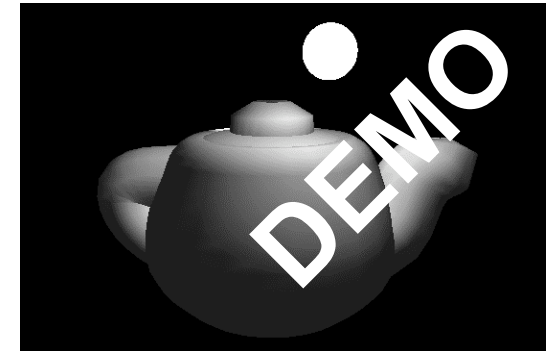


■ Vertex Shader: animated teapot

```
void main( // input
    float4 position      : POSITION, // position in object coordinates
    float3 normal        : NORMAL, // normal

    // user parameters
    uniform float4x4 objectMatrix, // object coordinate system matrix
    uniform float4x4 objectMatrixIT, // object coordinate system matrix inverse transpose
    uniform float4x4 modelViewMatrix, // modelview matrix
    uniform float4x4 modelViewMatrixIT, // modelview matrix inverse transpose
    uniform float4x4 projectionMatrix, // projection matrix
    uniform float deformation, // deformation parameter
    uniform float3 lightPosition, // light position
    uniform float3 lightAmbient, // light ambient parameter
    uniform float3 lightDiffuse, // light diffuse parameter
    uniform float3 lightSpecular, // light specular parameter
    uniform float3 lightAttenuation, // light attenuation parameter - constant, linear, quadratic
    uniform float3 materialEmission, // material emission parameter
    uniform float3 materialAmbient, // material ambient parameter
    uniform float3 materialDiffuse, // material diffuse parameter
    uniform float3 materialSpecular, // material specular parameter
    uniform float materialShininess, // material shininess parameter

    // output
    out float4 outPosition : POSITION, // position in clip space
    out float4 outColor    : COLOR ) // out color
{
```



Cg Example – Vertex Shader



```
// transform position from object space to clip space
float4 positionObject = mul(objectMatrix, position);

// transform normal into world space
float4 normalObject = mul(objectMatrixIT, float4(normal,1));
float4 normalWorld = mul(modelViewMatrixIT, normalObject);

// world position of light
float4 lightPositionWorld = \
    mul(modelViewMatrix, float4(lightPosition,1));

// assume viewer position is in origin
float4 viewerPositionWorld = float4(0.0, 0.0, 0.0, 1.0);

// apply deformation
positionObject.xyz = positionObject.xyz + \
    deformation * normalize(normalObject.xyz);
float4 positionWorld = mul(modelViewMatrix, positionObject);
outPosition = mul(projectionMatrix, positionWorld);

// two vectors
float3 P = positionWorld.xyz;
float3 N = normalize(normalWorld.xyz);

// compute the ambient term
float3 ambient = materialAmbient*lightAmbient;

// compute the diffuse term
float3 L = normalize(lightPositionWorld.xyz - P);
float diffuseFactor = max(dot(N, L), 0);
float3 diffuse = materialDiffuse * lightDiffuse * diffuseFactor;
```

```
// compute the specular term
float3 V = normalize( viewerPositionWorld.xyz - \
    positionWorld.xyz);
float3 H = normalize(L + V);
float specularFactor = \
    pow(max(dot(N, H), 0), materialShininess);
if (diffuseFactor <= 0) specularFactor = 0;
float3 specular = \
    materialSpecular * \
    lightSpecular * \
    specularFactor;

// attenuation factor
float distanceLightVertex = \
    length(P-lightPositionWorld.xyz);
float attenuationFactor = \
    1 / ( lightAttenuation.x + \
    distanceLightVertex*lightAttenuation.y + \
    distanceLightVertex*distanceLightVertex*\
    lightAttenuation.z );

// set output color
outColor.rgb = materialEmission + \
    ambient + \
    attenuationFactor * \
    ( diffuse + specular );
outColor.w = 1;
}
```

Cg Example – Phong Shading



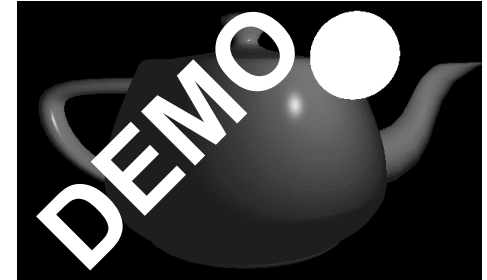
vertex shader

```
void main( float4 position      : POSITION, // position in object coordinates
           float3 normal       : NORMAL,  // normal

           // user parameters
           ...

           // output
           out float4 outTexCoord0 : TEXCOORD0, // world normal
           out float4 outTexCoord1 : TEXCOORD1, // world position
           out float4 outTexCoord2 : TEXCOORD2, // world light position
           out float4 outPosition  : POSITION)    // position in clip space
{
    // transform position from object space to clip space
    ...
    // transform normal into world space
    ...

    // set world normal as out texture coordinate0
    outTexCoord0 = normalWorld;
    // set world position as out texture coordinate1
    outTexCoord1 = positionWorld;
    // world position of light
    outTexCoord2 = mul(modelViewMatrix, float4(lightPosition,1));
}
```



Cg Example – Phong Shading



fragment shader

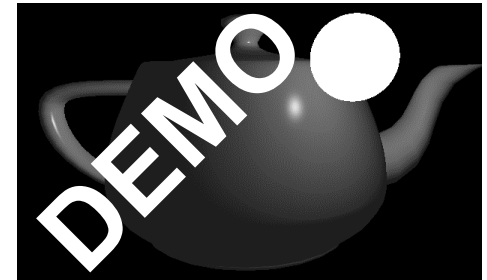
```
void main( float4    normal    : TEXCOORD0,    // normal
           float4    position  : TEXCOORD1,    // position
           float4    lightPosition : TEXCOORD2, // light position
           out float4 outColor   : COLOR )
{
    // compute the ambient term
    ...

    // compute the diffuse term
    ...

    // compute the specular term
    ...

    // attenuation factor
    ...

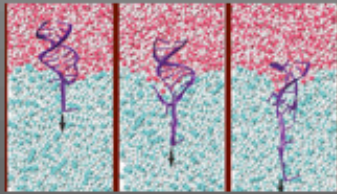
    // set output color
    outColor.rgb = materialEmission + ambient + attenuationFactor * (diffuse + specular);
}
```



- general purpose computation on the GPU
- in the past: access via shading languages and rendering pipeline
- now: access via cuda interface in C environment



GPGPU Applications

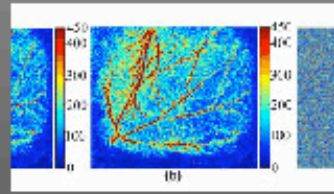


Scalable Molecular Dynamics: NAMD



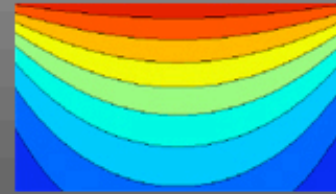
Flame Fractals

50 x



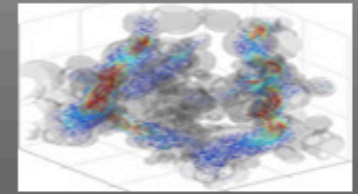
Fast Blood Flow Visualization of High-resolution Laser Speckle Imaging Data

60 x



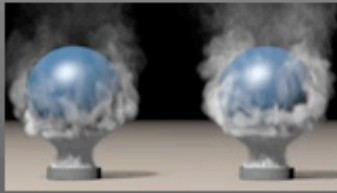
Computational Fluid Dynamics (CFD) using GPUs

17 x



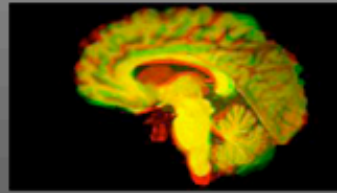
Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphi

28 x



Low Viscosity Flow Simulations for Animation

55 x



Accelerated Image Registration With CUDA

100 x



Ray tracing with CUDA (CUDART-sp)

25 x



UT3 PhysX Mod Pack Using CUDA

5 x



GPU Acceleration Solutions

35 x



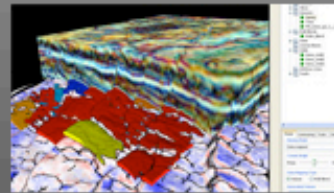
GpuCV: GPU-accelerated Computer vision library

100 x



Fast Computed Tomography

50 x

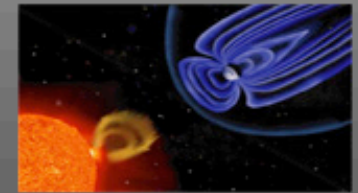


SVI Pro Advanced 3D Seismic Analysis

34 x



Glimmer: Multilevel MDS on the GPU



GPU Particle Tracking and Multi-Fluid Simulations with Greatly Enhanced

50 x

[courtesy NVIDIA]