Emulating an Offline Renderer by 3D Graphics Hardware

Jörn Loviscach
Hochschule Bremen, Germany

Outline

- Objectives
- .fx files
- Implementation
- Performance
- Summary and Outlook

Objectives

- Leverage the computing power of graphics cards for interactive, but high-quality preview and final movie rendering.
- Allow .fx materials to be used in final renderings (large frame sizes, movie files; shadows, antialiasing).

cf. Alias Maya: Hardware Renderer, Conv

.fx Files

Entire programming of a material contained in one text file:

- vertex and pixel shaders
- render settings (alpha etc.)
- a number of techniques
- a number of rendering passes
- specification of inputs (matrices etc.)
- parameters with GUI

Toolkits: Microsoft DirectX 9, Nvidia CgFX

.fx Files

Original purpose:
Let game designers
use complex shaders.

in 3D software (Maya, XSI, 3ds max):

- construct models
- assign .fx materials
- adjust with real-time preview

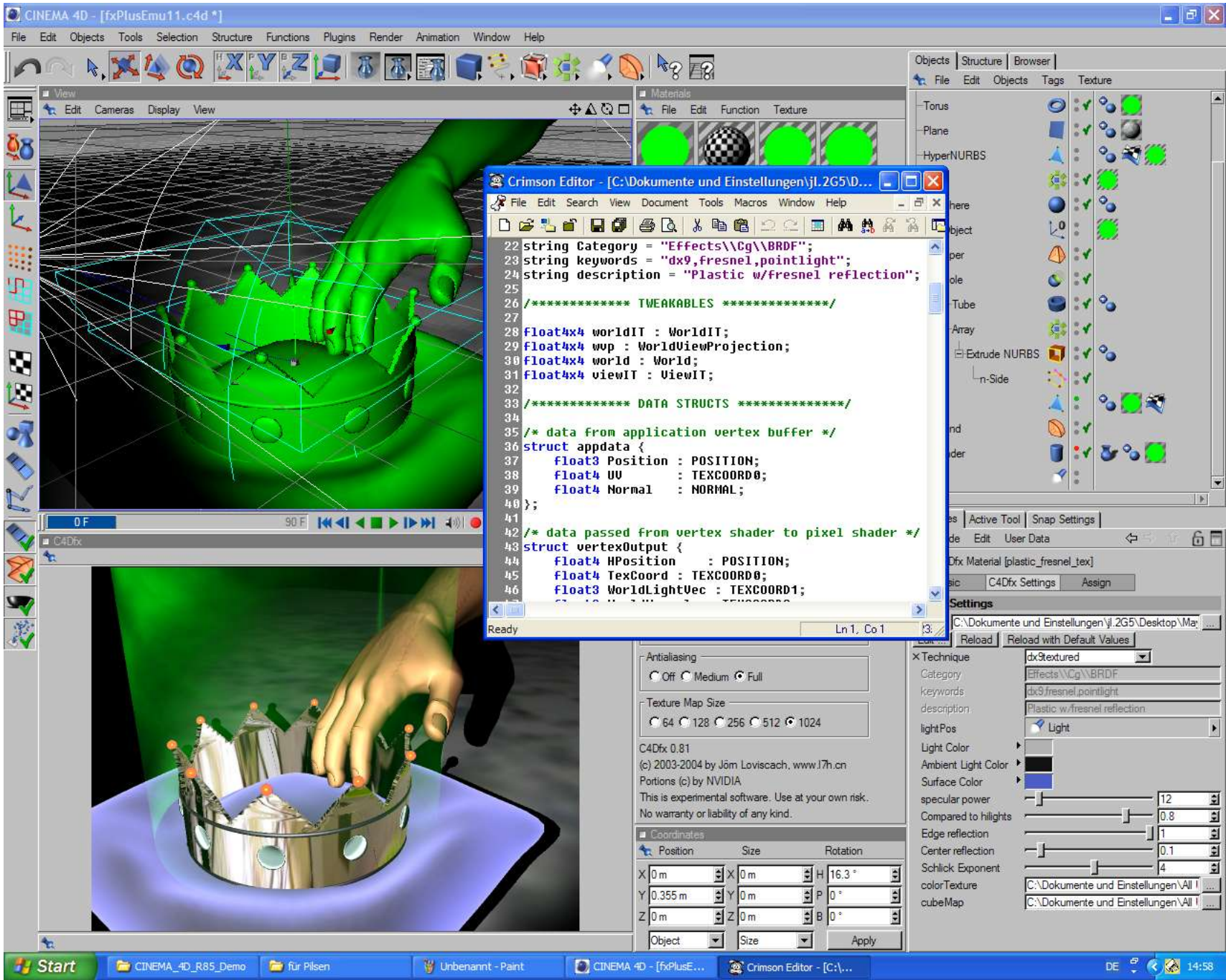
in game engine:

- load and display models

Implementation

- plug-in for Cinema 4D
- based on OpenGL and CgFX
- load, display, and edit .fx files
- GUI built via API of Cinema 4D

Implementation



Implementation

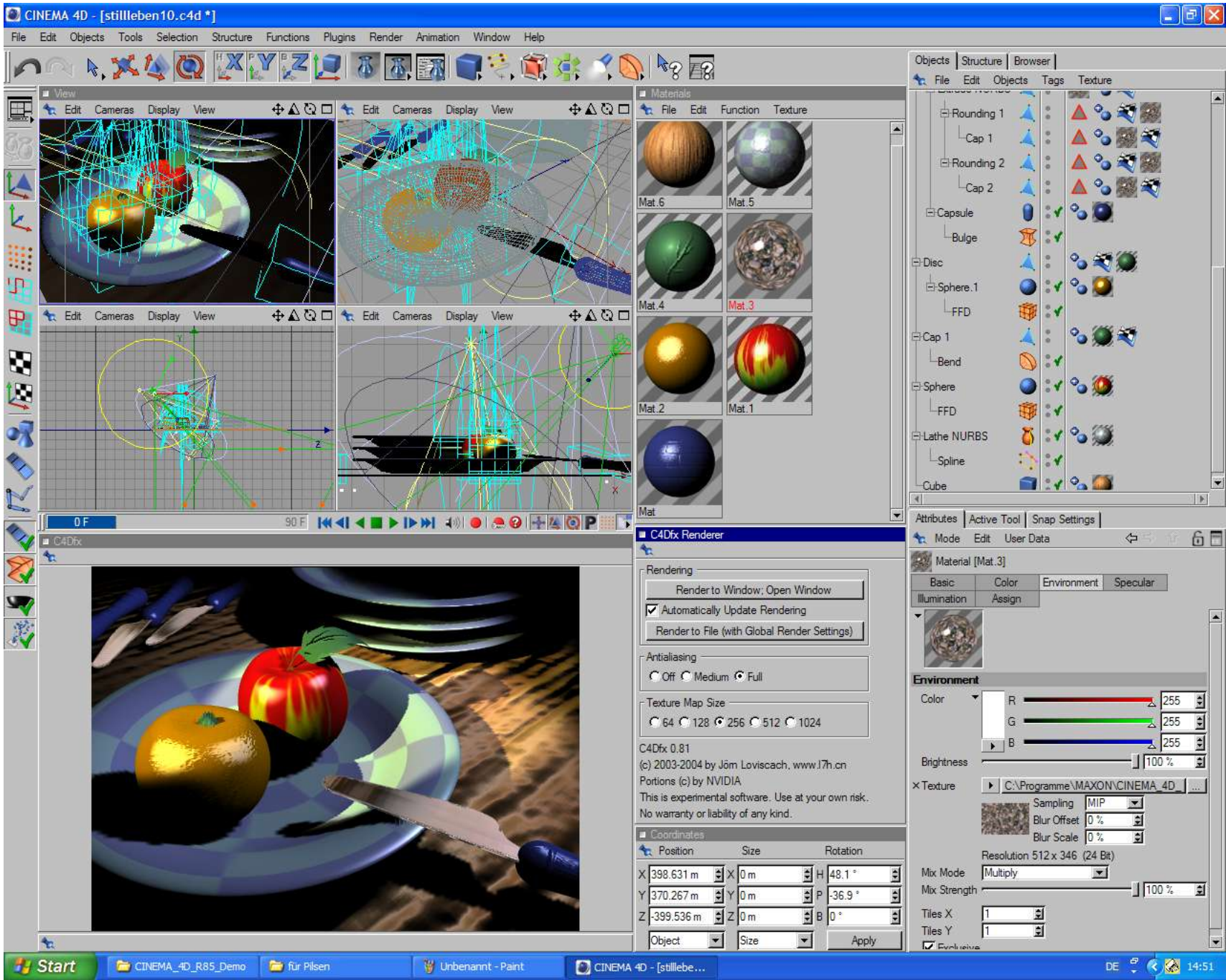
- Convert standard materials to memory-based .fx text strings

```
float4x4 wvp;
float4x4 wit;
float4x4 w;
float4x4 vit;
float4 diffCol;
float4 lumiCol;
texture diffuseTexture;
float bumpHeight;
texture normalTexture;
float4 envCol;
texture envTexture;
texture specShapeTexture;
float4 specCol;
float4 light0Pos0;
float4 light0Col0;
float4 light0Params0;
float4 light0Up0;
float4 light0Dir0;
float4 light0Side0;
float4 light0Pos1;
float4 light0Col1;
float4 light0Params1;
float4 light0Up1;
float4 light0Dir1;
float4 light0Side1;
texture depthTexture0;
texture depthTexture1;
sampler2D diffuseSampler = sampler_state
{   Texture = <diffuseTexture>;
    MinFilter = Linear;
    MagFilter = Linear;
    MipFilter = Linear;
};
samplerCUBE specShapeSampler = sampler_state
{   Texture = <specShapeTexture>;
    AddressU = Clamp;
    MinFilter = Linear;
    MagFilter = Linear;
    MipFilter = None;
};
sampler2D normalSampler = sampler_state
{   Texture = <normalTexture>;
    MinFilter = Linear;
    MagFilter = Linear;
    MipFilter = Linear;
};
samplerCUBE envMapSampler = sampler_state
{   Texture = <envTexture>;
    MinFilter = Linear;
    MagFilter = Linear;
    MipFilter = Linear;
};
sampler2D depthSampler0 = sampler_state
{   Texture = <depthTexture0>;
    MinFilter = Linear;
    MagFilter = Linear;
    MipFilter = None;
    AddressU = Border;
    AddressV = Border;
    BorderColor = {0.0f, 0.0f, 0.0f, 1.0f};
};
sampler2D depthSampler1 = sampler_state
{   Texture = <depthTexture1>;
    MinFilter = Linear;
    MagFilter = Linear;
    MipFilter = None;
    AddressU = Border;
    AddressV = Border;
    BorderColor = {0.0f, 0.0f, 0.0f, 1.0f};
};
struct appdata {
    float4 position   : POSITION;
    float4 norm : NORMAL;
    float4 uv: TEXCOORD0;
    float4 tang : TEXCOORD1;
    float4 binorm : TEXCOORD2;
};
struct vertexOutput {
    float4 hPos : POSITION;
    float4 uv : TEXCOORD0;
    float3 norm : TEXCOORD1;
    float3 tang : TEXCOORD2;
    float3 binorm : TEXCOORD3;
    float3 view : TEXCOORD4;
    float3 wPos : TEXCOORD5;
};
struct pixelOutput {
    float3 col : COLOR;
};
vertexOutput mainVS(appdata IN,
    uniform float4x4 wvp,
    uniform float4x4 wit,
    uniform float4x4 w,
    uniform float4x4 vit)
{
    vertexOutput OUT;
    OUT.uv = IN.uv;
    OUT.hPos = mul(wvp, IN.position);
    OUT.norm = mul(wit, IN.norm).xyz;
    OUT.tang = mul(w, IN.tang).xyz;
    OUT.binorm = mul(w, IN.binorm).xyz;
    float3 pW = mul(w, IN.position).xyz;
    OUT.wPos = pW;
    OUT.view = normalize(pW - vit[3].xyz);
    return OUT;
}
pixelOutput mainPS(vertexOutput IN,
    uniform sampler2D diffuseSampler,
    uniform samplerCUBE specShapeSampler,
    uniform sampler2D normalSampler,
    uniform samplerCUBE envSampler,
    uniform sampler2D depthSampler0,
    uniform sampler2D depthSampler1,
    uniform float4 lumiCol,
    uniform float4 diffCol,
    uniform float bumpHeight,
    uniform float4 envCol,
    uniform float4 specCol,
    uniform float4 light0Pos0,
    uniform float4 light0Col0,
    uniform float4 light0Params0,
    uniform float4 light0Up0,
    uniform float4 light0Dir0,
    uniform float4 light0Side0,
    uniform float4 light0Pos1,
    uniform float4 light0Col1,
    uniform float4 light0Params1,
    uniform float4 light0Up1,
    uniform float4 light0Dir1,
    uniform float4 light0Side1)
{
    pixelOutput OUT;
    float3 Vn = normalize(IN.view);
    float3 Nn = normalize(IN.norm);
    float3 tangn = normalize(IN.tang);
    float3 binorm = normalize(IN.binorm);
    float2 bumps = bumpHeight*(tex2D(normalSampler, IN.uv.xy).xy * 2.0 - float2(1.0, 1.0));
    float3 Nb = normalize(bumps.x*tangn + bumps.y*binorm + Nn);
    float3 env = texCUBE(envSampler, reflect(Vn, Nb)).rgb;
    float3 colorSum = lumiCol.rgb + env*envCol.rgb;
    float3 baseDiffCol = diffCol.rgb + tex2D(diffuseSampler, IN.uv.xy).rgb;
    {
        float3 Ld = light0Pos0.xyz - IN.wPos;
        float3 Ln = normalize(Ld);
        float3 baseCol = max(0.0, dot(Ln, Nb))*baseDiffCol;
        float spec = tex1D(specShapeSampler, dot(Vn, reflect(Ln, Nb))).r;
        baseCol += specCol.rgb*spec;
        float3 Ll = (Ln/dot(Ln, light0Dir0.xyz) - light0Dir0.xyz)*light0Params0.z;
        float shadowFactor = max(light0Params0.x, smoothstep(1.0, light0Params0.w, length(Ll)));
        float d = dot(Ld, light0Dir0.xyz);
        float z = 10.1010101/d + 1.01010101;
        float2 depthUV = float2(0.5, 0.5) + 0.5*float2(dot(Ll, light0Side0.xyz), dot(Ll, light0Up0.xyz));
        shadowFactor *= max(light0Params0.y, tex2Dproj(depthSampler0, float4(depthUV.x, depthUV.y, z-0.0002, 1.0)).x);
        colorSum += shadowFactor*baseCol*light0Col0.rgb;
    }
    {
        float3 Ld = light0Pos1.xyz - IN.wPos;
        float3 Ln = normalize(Ld);
        float3 baseCol = max(0.0, dot(Ln, Nb))*baseDiffCol;
        float spec = tex1D(specShapeSampler, dot(Vn, reflect(Ln, Nb))).r;
        baseCol += specCol.rgb*spec;
        float3 Ll = (Ln/dot(Ln, light0Dir1.xyz) - light0Dir1.xyz)*light0Params1.z;
        float shadowFactor = max(light0Params1.x, smoothstep(1.0, light0Params1.w, length(Ll)));
        float d = dot(Ld, light0Dir1.xyz);
        float z = 10.1010101/d + 1.01010101;
        float2 depthUV = float2(0.5, 0.5) + 0.5*float2(dot(Ll, light0Side1.xyz), dot(Ll, light0Up1.xyz));
        shadowFactor *= max(light0Params1.y, tex2Dproj(depthSampler1, float4(depthUV.x, depthUV.y, z-0.0002, 1.0)).x);
        colorSum += shadowFactor*baseCol*light0Col.rgb;
    }
    OUT.col = colorSum;
    return OUT;
}
technique t0
{
    pass p0
    {
        VertexShader = compile vs_2_x mainVS(wvp, wit, w, vit);
        ZEnable = true;
        ZWriteEnable = true;
        CullMode = None;
        PixelShader = compile ps_2_x mainPS(diffuseSampler, specShapeSampler, normalSampler, envMapSampler,
            depthSampler0,
            depthSampler1,
            lumiCol, diffCol, bumpHeight, envCol, specCol,
            light0Pos0, light0Col0, light0Params0, light0Up0, light0Dir0, light0Side0,
            light0Pos1, light0Col1, light0Params1, light0Up1, light0Dir1, light0Side1);
    }
}
```

Implementation

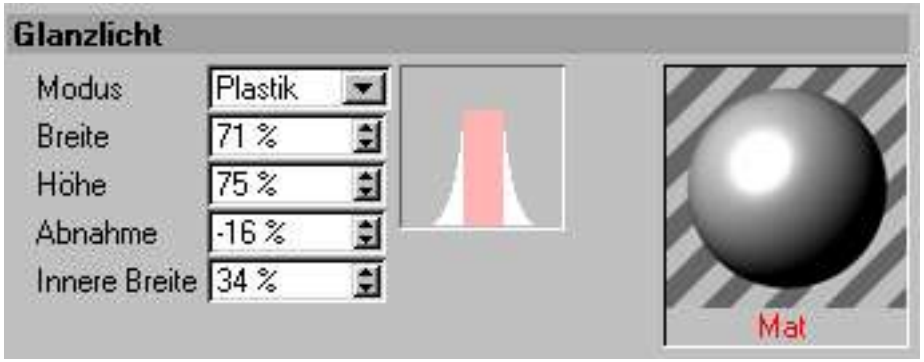
- Convert color textures
- Convert environments to cube maps
- Convert bump maps to normal maps

Implementation



Implementation

- Render via invisible Win32 threads working in offscreen buffers
- Compute shadow maps
- Convert complex highlight shape to 1D texture



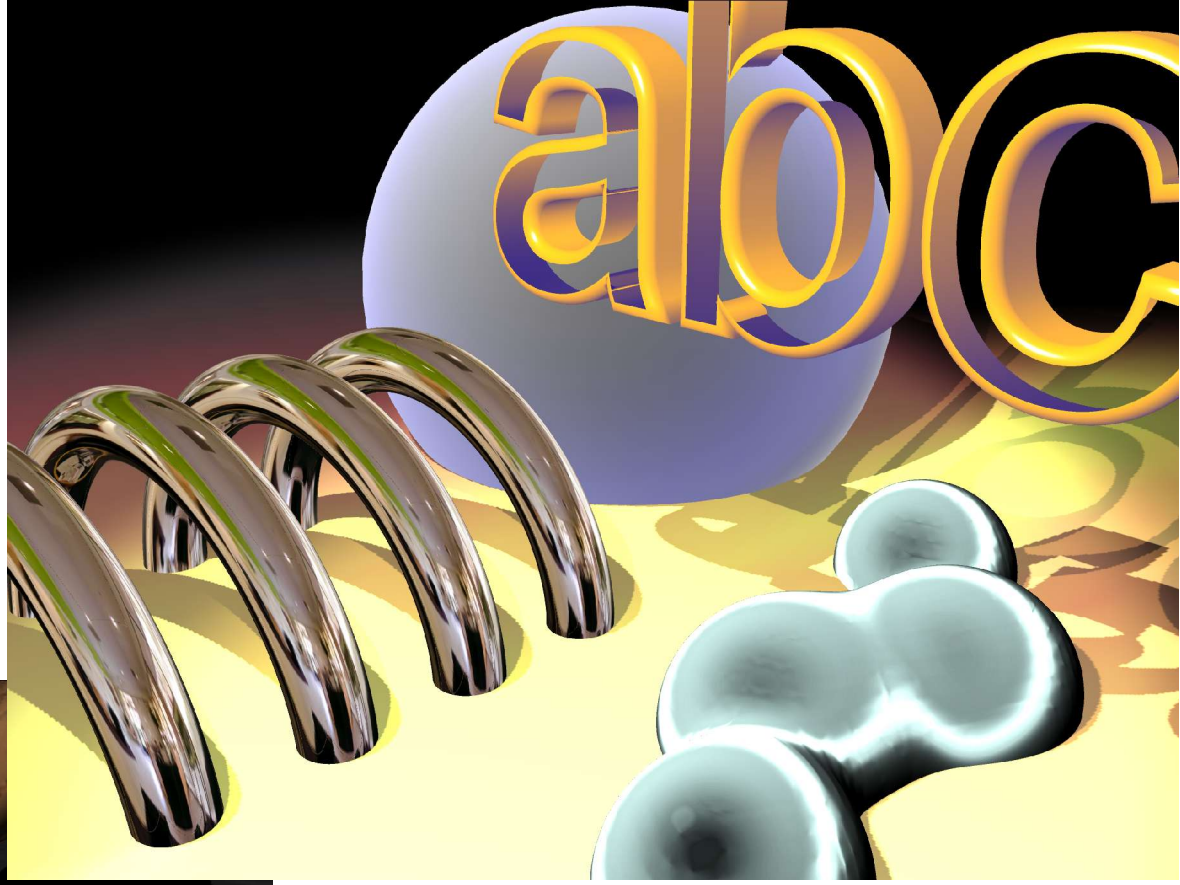
Performance

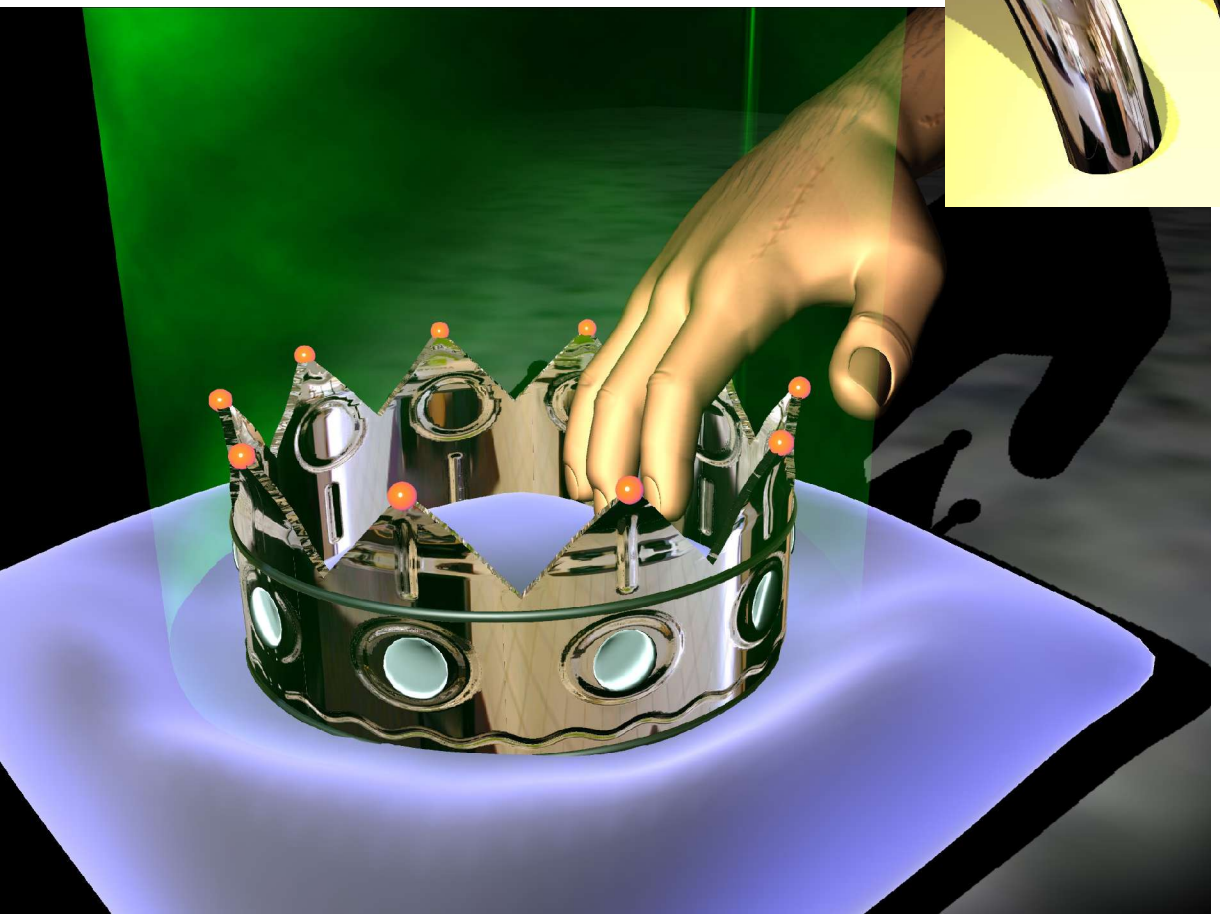
Hardware

Software

Performance

.fx materials
casting shadows





large format,
e.g., 2048 x 1536

Performance

Visual quality:

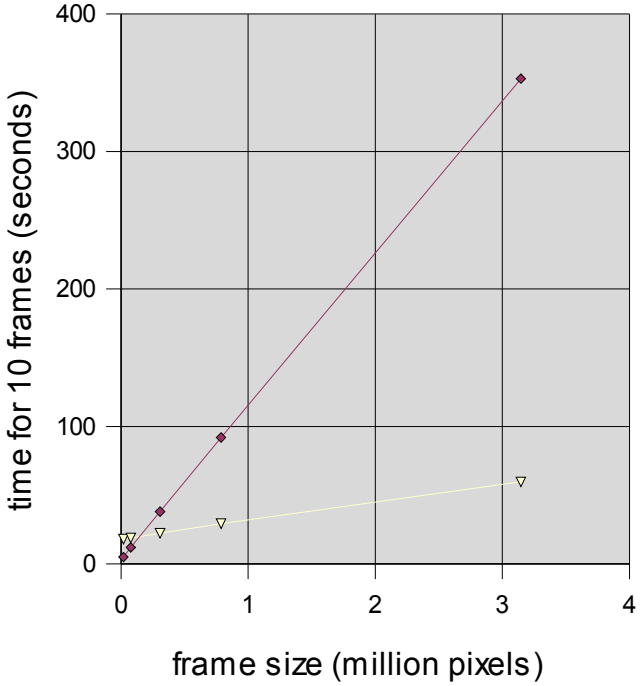
- lighting, specular highlighs, lateral dropoff of cone light: ok
- color textures: ok
- environment maps: ok
- bump maps: look different (exact computation scheme not available)
- shadows: hard and jagged
- reflection, refraction: none

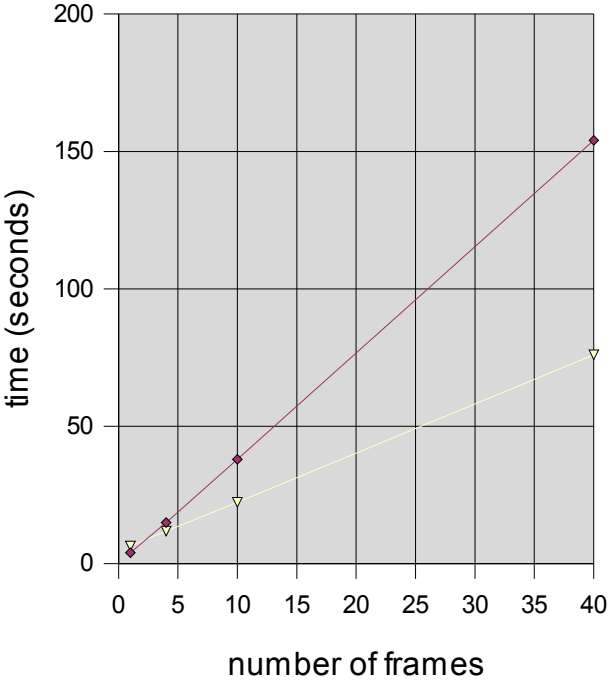
Performance

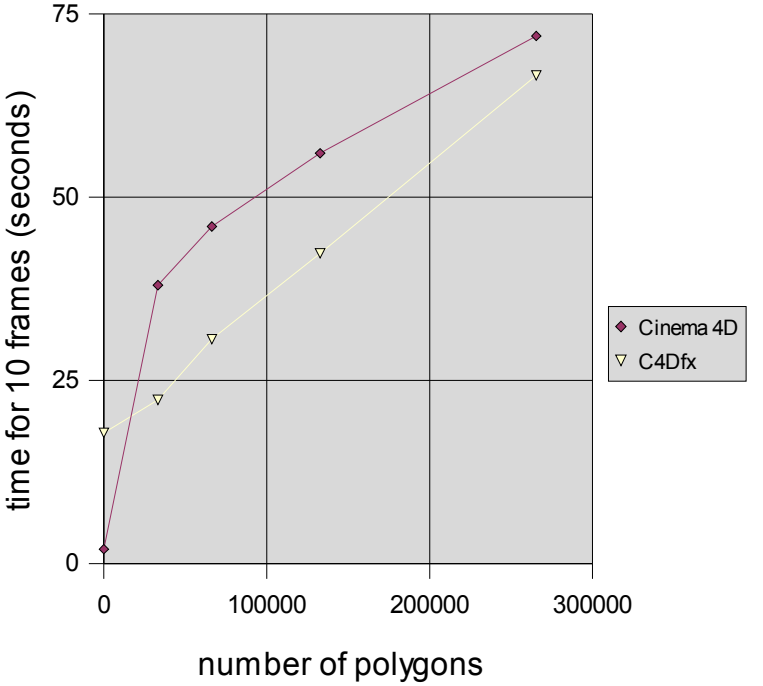
Benchmark

Base line: 33.000 polygons, 10 frames, 640 x 480 pixels
System: Pentium-4 @ 2.5 GHz, Nvidia GeForce FX 5900

Performance







- Dramatic speed-up concerning pixels/s
- But first pixel and first frame are expensive.
- Optimization for larger scenes?

Summary, Outlook

- Plug-in for Cinema 4D combining .fx display and hardware rendering
- Impressive speed-up possible for long films, large frame sizes, not-too-complex scenes
- Optimization: Do not re-compute textures/objects more often than necessary.
- Optimization: Occlusion culling etc.