Shader Programming: An Introduction Using the Effect Framework

Jörn Loviscach
jlovisca@informatik.hs-bremen.de

Hochschule Bremen
University of Applied Sciences
Bremen, Germany

Agenda

- A First Glance at Shader Programming
- Review of Basic 3D Techniques with .fx
- Phong Illumination Model and Interpolation

- Break

- Basic Shader Effects
- Bump Mapping
- Complex Shader Effects
- Outlook

A First Glance at Shader Programming

Examples for Shaders

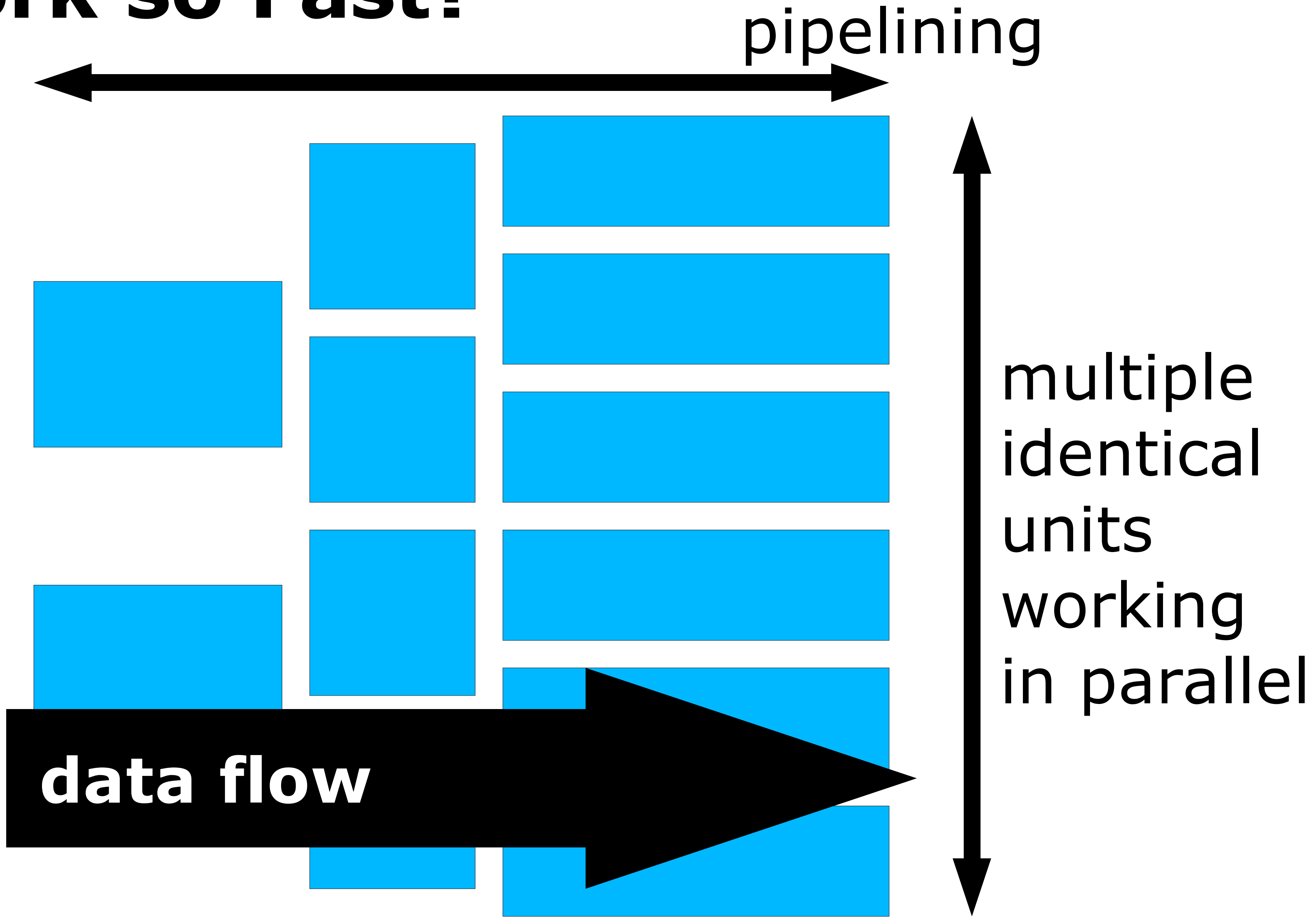
Typical:

- Phong Interpolation
- Deformation
- Bump-mapping

Not so typical:

- Glow (frame-based)
- Glow (pseudo-geometry)
- Shadow Volume Extrusion

Why Can Graphics Cards Work so Fast?



Why Can Graphics Cards Work so Fast?

parallel processing -->
restrictions in programming model

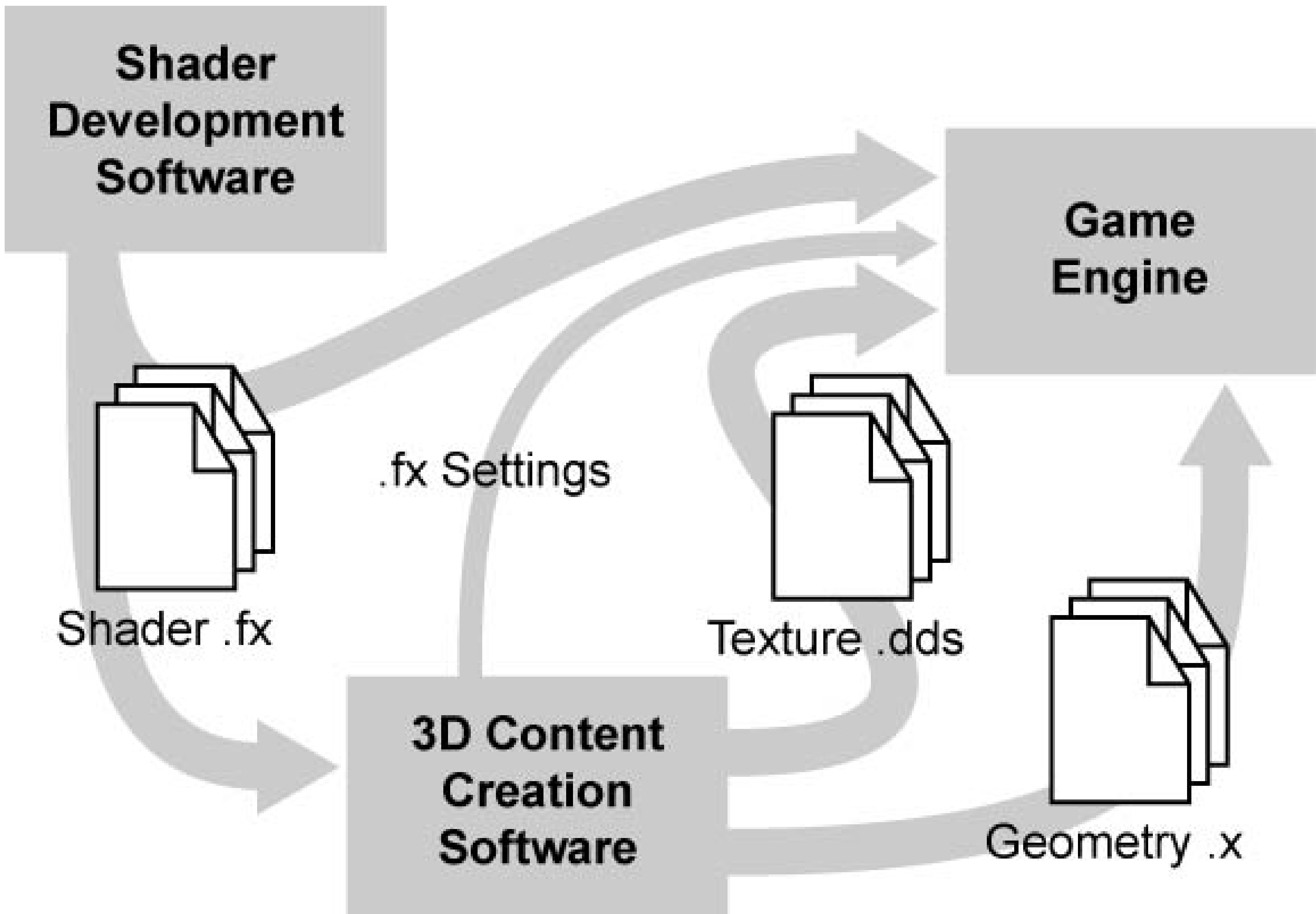
specialized units -->
(only?) special functionality available

Shading Languages

- Assembler languages

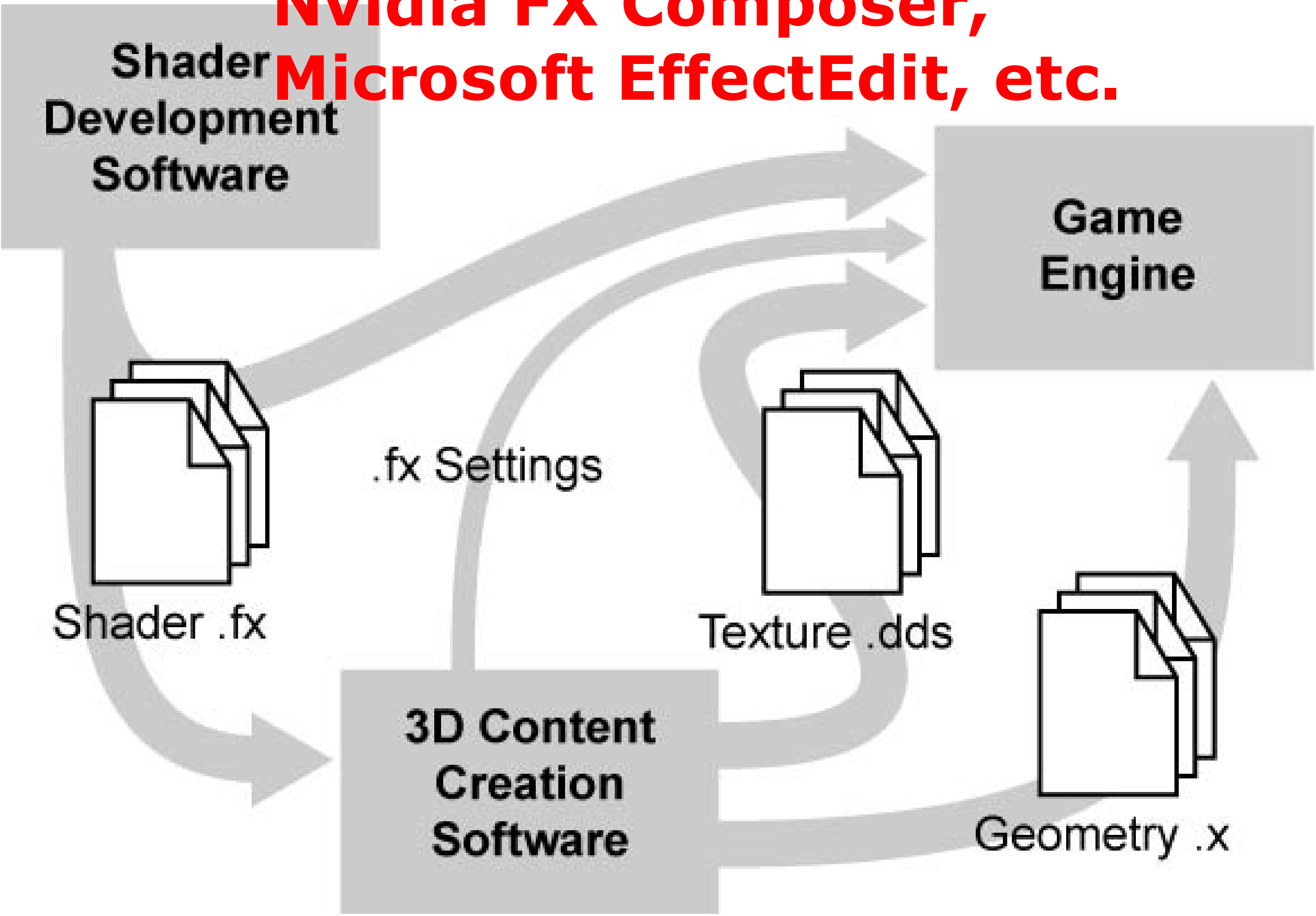
- High-level languages
  - (RenderMan Shading Language)
  - Nvidia Cg $\approx$ Microsoft HLSL
  - OpenGL Shading Language

The Effect Framework

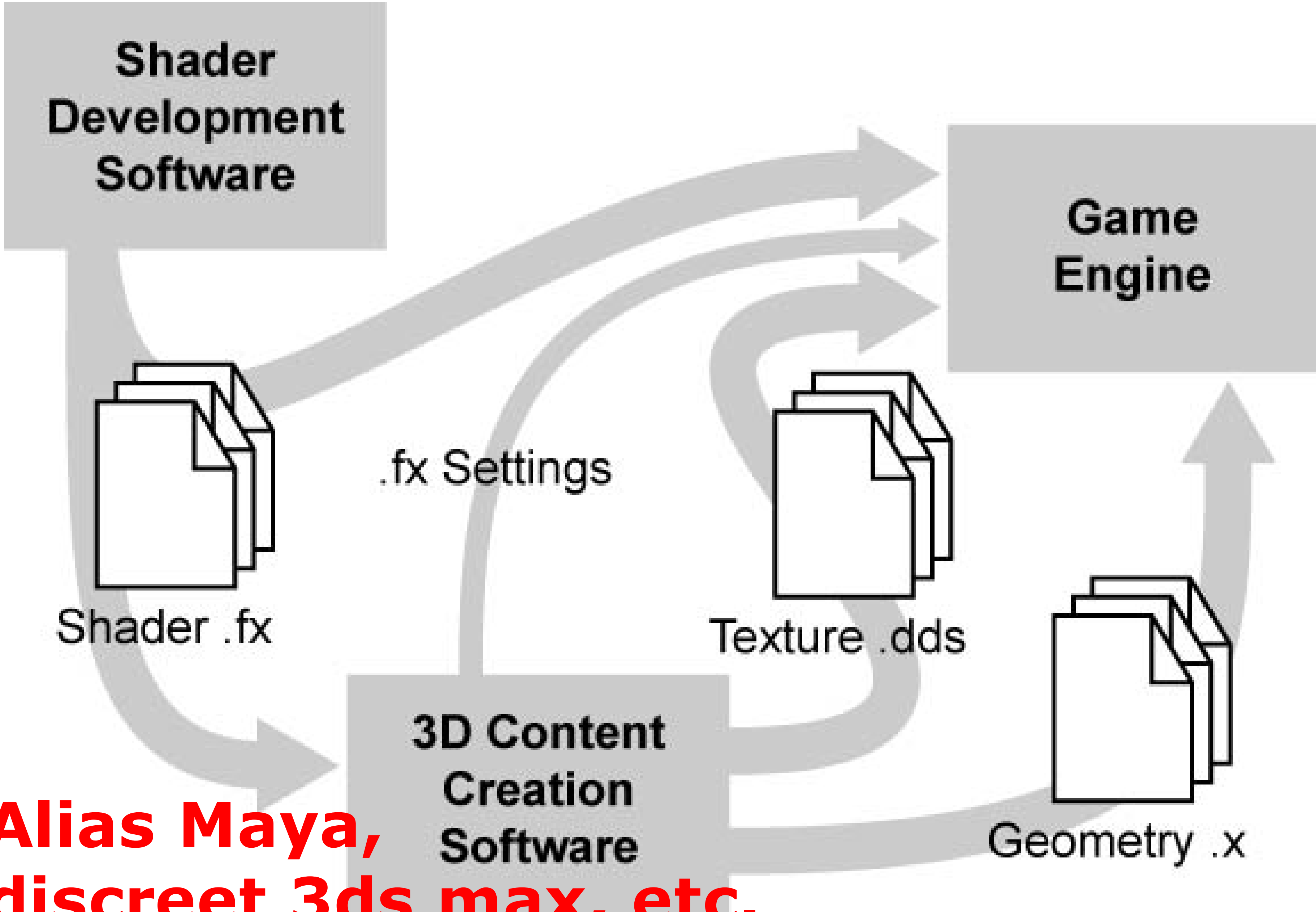


The Effect Framework

Nvidia FX Composer, Microsoft EffectEdit, etc.



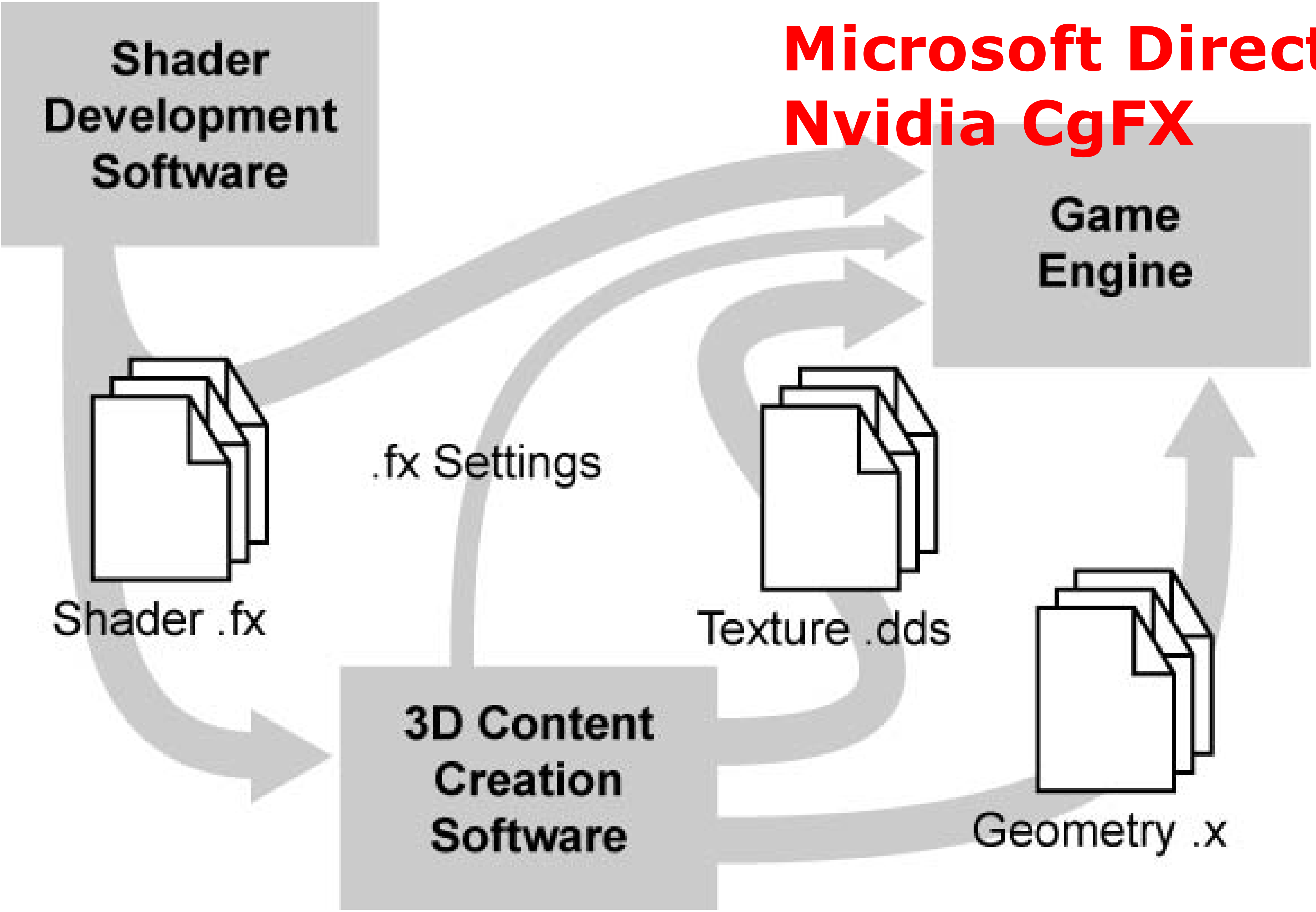
The Effect Framework



Alias Maya,
discreet 3ds max, etc.

The Effect Framework

Microsoft DirectX, Nvidia CgFX



Review of Basic 3D Techniques with .fx

Four-Component Vectors

(Open jl_simplematerial.fxproj.)

Point: (px, py, pz, 1)
Vector (direction+length):
(vx, vy, vz, 0)
Color: (r, g, b, a)

- Color range: 0.0 to 1.0
- Precision: **`float`** and **`half`**
- **`float3`** etc.
- Swizzling and masking

Transformations and Homogeneous Coordinates 1

- Perspective Transforms cannot be written with matrices as usual.
- Trick: 4x4 matrix, perspective divide

Matrix

- (x,y,z,w) ---> (x',y',z',w') ---> (x',y',z')/w'
- Compare: foreshortening
- Rotation, scaling, linear perspective, and translation represented by 4x4 matrices
- Homogenous: common factor cancels
- Translation affects points, but not vectors

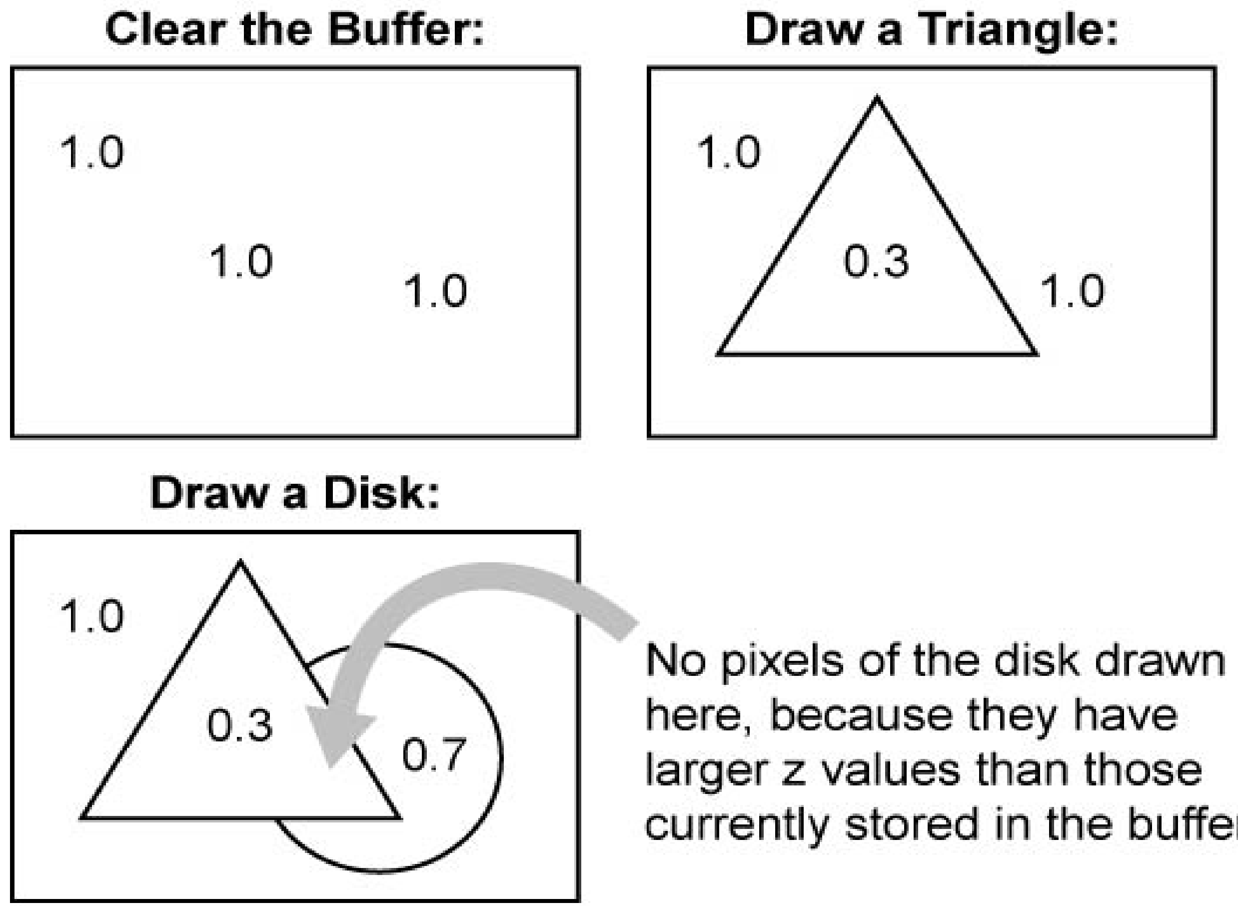
Transformations and Homogeneous Coordinates 2

- DirectX uses row vectors,
  not column vectors by default:
  Multiply vector * matrix
- Composition of transformations:
  (((v*M1)*M2)*M3 = v * (M1*M2*M3),
  Reduction to one single product involving v
- Standard matrices in DirectX:
  World: Position and orient an object
  View: Position and orient the camera
  Projection: Choose the camera's lense

Back Face Culling, z-Buffer

Back Face Culling helps with visibility only for closed convex objects, but improves speed for all closed objects.

z-Buffer:
standard
real-time
solution
for visibility
computation



Real-Time Rendering Pipeline

"Fixed-function"

- Transform and Lighting
- Perspective Divide
- Triangle Setup and Rasterization
- Shading and Texturing
- Depth Test
- Alpha Blending

Real-Time Rendering Pipeline

"Fixed-function"

- Transform and Lighting
- Perspective Divide
- Triangle Setup and Rasterization
- Shading and Texturing
- Depth Test
- Alpha Blending

"Programmable"

- **Vertex Shader** (Vertex Program)
- Perspective Divide
- Triangle Setup and Rasterization
- **Pixel Shader** (Fragment Program)
- Depth Test
- Alpha Blending

Restrictions to Shaders

Vertex Shaders:

- Access to only one vertex
- Must set position
- Vertex may not be duplicated or deleted (but may e.g. be moved outside the view)
- No access to textures (different in SM 3.0)

Pixel Shaders:

- Access to only one pixel
- Must set color
- Access to screen-space differences
- Screen position fixed
- Pixel may be discarded (clipped)
- Access to textures

Phong Illumination Model and Interpolation

Illumination: Normals 1

(Open jl_phong.fxproj.)

- Lighting depends (mostly) on the angle between the local tangent plane to the object and the light source.
- Tangent plane hard to compute based on points.
- Solution: Equip each vertex with a normal vector (mostly, of unit length).

- "Semantics" POSITION and NORMAL in HLSL

Illumination: Normals 2

- Normals given in object space,
  but lighting computed in world space:
  Conversion?
- Only translation or rotation:
  Use World matrix
- Uniform scaling contained, too:
  Use World matrix and normalize afterwards
- General case:
  $(M\mathbf{a}) \times (M\mathbf{b}) = \det(M)\,(M^{-1})^T(\mathbf{a} \times \mathbf{b})$
  Thus use WorldInverseTranspose
  as transformation matrix for normals;
  normalize afterwards.

Parameters and Annotations

Semantic

```
float4 DiffuseColor : Diffuse
<
  string UIName = "Diffuse Color";
> = {0.6, 0.9, 0.6, 1.0};
```

Default Value

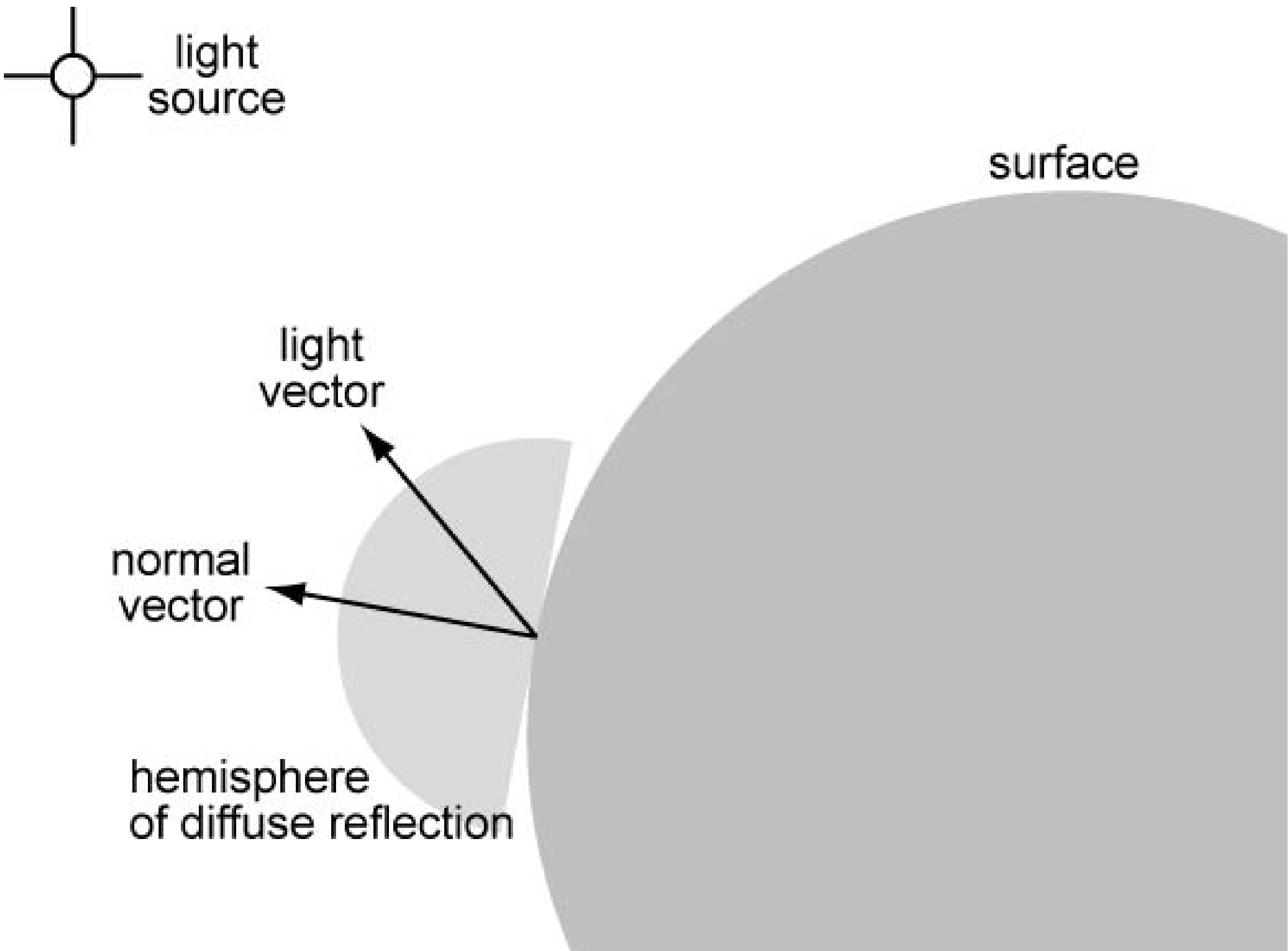
Annotation

```
float4 LightPosition : Position
<
  string Object = "PointLight";
  string Space = "World";
> = {-1.0, 2.0, 1.0, 1.0};
```

- Connecting Parameters to UI Elements

Phong Illumination in the Vertex Shader 1

Phong illumination =
constant
+ **diffuse**
+ specular

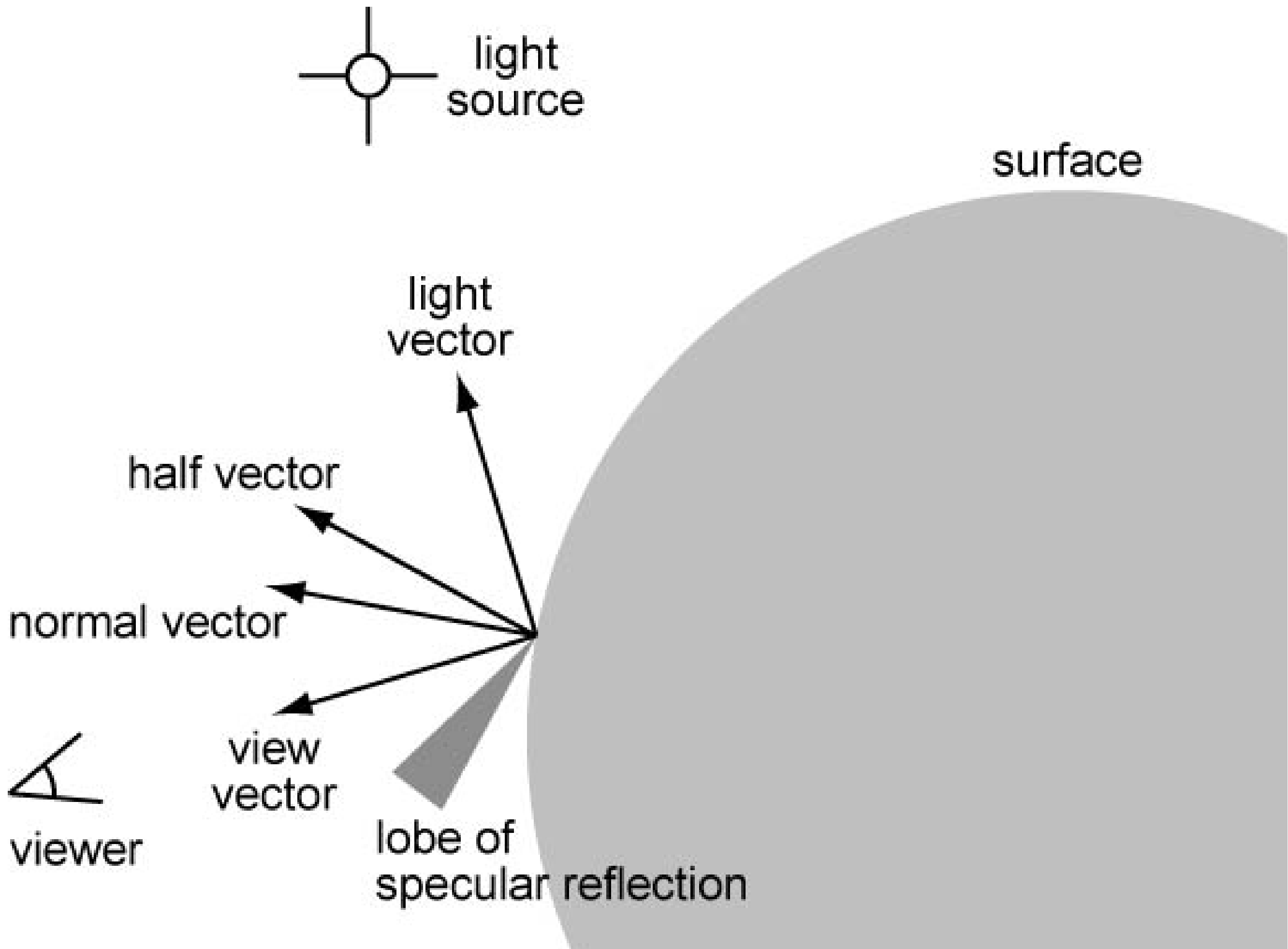


Phong Illumination in the Vertex Shader 2

Phong illumination =
  constant
+ diffuse
+ **specular**

Viewer position:
`VI[3].xyz`

`lit` function



Phong Interpolation

Data transfer from vertex to pixel shader:

```
struct VertexOutput
{
  float4 HP : POSITION; // homog.
  float3 N : TEXCOORD0; // normal
  float3 V : TEXCOORD1; // to viewer
  float3 L : TEXCOORD2; // to light
};
```

- Position may not be read in pixel shader.
- All values interpolated between vertices.
- **`TEXCOORD`***n* to transfer unclamped values.

Phong Illumination in the Pixel Shader

- Vectors needed in the computation: normal, view vector, light vector.
- These may be computed per vertex (precise enough if no bump mapping; computation per pixel incurs higher costs).
- Automatic interpolation computes per-pixel vectors.
- Interpolation denormalizes vectors; may need normalization in pixel shader.

Basic Shader Effects

Deformation

(Open jl_deformation.fxproj.)

Subject the position **x** to a mapping **x** -> **f** (**x**) in the vertex shader.

But: Normals have to change, too. Use inverse transposed Jacobian matrix.

Texture Mapping

(Open jl_texture.fxproj.)

- Textures: Putting wallpaper onto 3D surfaces

```
texture DiffuseTexture : Diffuse
//...
sampler DiffuseMap = sampler_state
//...
float4 t = tex2D(DiffuseMap, IN.UV);
```

- Deforming uv space
- Creating textures in .dds format

Bump Mapping

Bump Mapping: Tangent Space

- Bump Mapping: Do not actually deform geometry, only use distorted normals.
- Store normal vectors in a texture;
- most efficient and easily controllable in locally adapted coordinate frame.
- Host application has to deliver unit vectors of that frame per vertex: normal, tangent, binormal
- Typically converted to `World` space in the vertex shader.

Bump Mapping: Normal Maps and Environment Maps

- Distorted normal (nx,ny,nz) stored in texture (normal map) as pseudo-RGB.
- Difficult to paint. Start with bump map instead and convert to normal map through gradient.

Environment map:

- Simulation (quite imprecise!) of perfect reflection
- Cube map = wallpaper put onto the inside of an infinitely large cube

Complex Shader Effects

Textures as Functions

- reduce computational load
- generate complex (life-like?) looks
- clipping/wrapping built-in

(Open jl_textures_as_functions.fxproj.)
Generalization of Phong lighting:

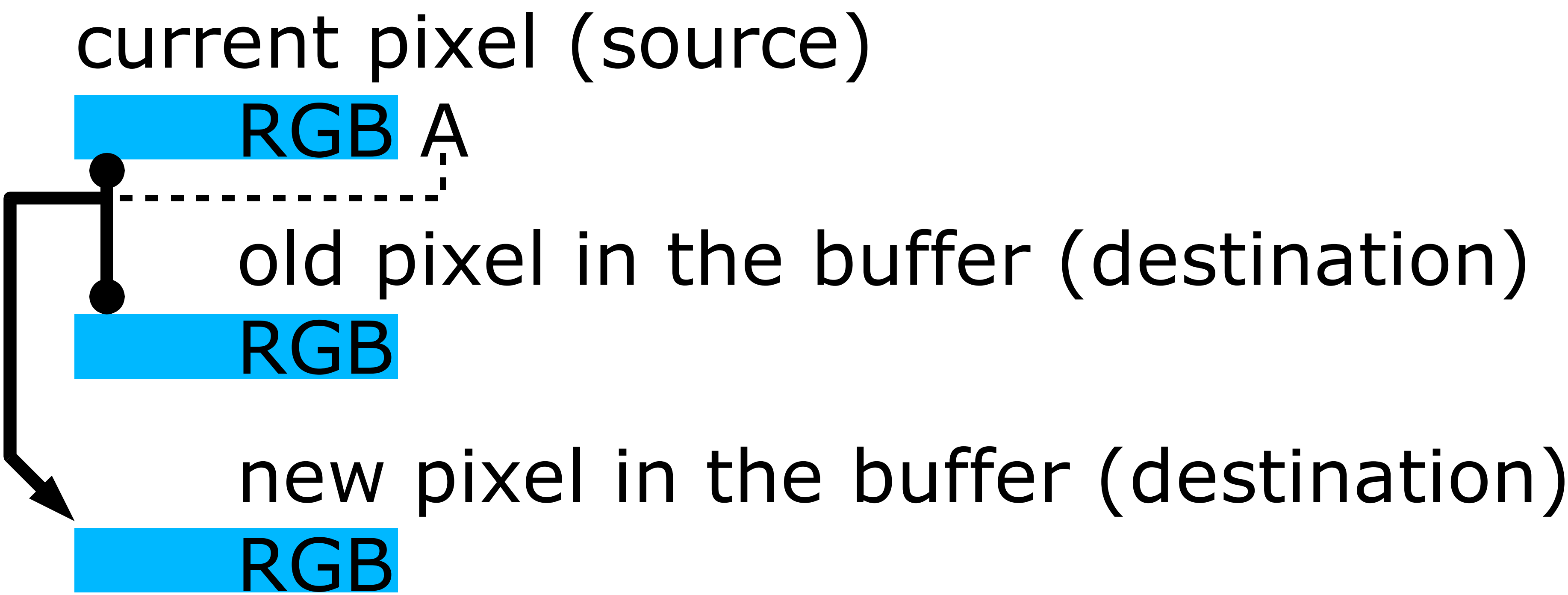
```
tex2D(LightingModel, float2(LdotN, HdotN));
```

(Open jl_textures_as_functions_2.fxproj.)
Versatile anisotropic reflection:

```
tex2D(HighlightModel,
0.5 * float2(HdotT, HdotB) + float2(0.5, 0.5));
```

Alpha Blending

(Open jl_alpha_material.fxproj.)



- Blending operations may also be configured differently.
- Drawing order affects transparency.

Multiple Rendering Passes

- Usage 1: Add different contributions, e.g., from several light sources. Problems: repetition of upfront computations; precision loss
- Usage 2: Deform geometry differently for each pass, e.g., for object and halo.

Outlook

Outlook 1

Using .fx in one's own software:
Toolkits: DirectX 9.0 or CgFX
Compute matrices etc. yourself
and hand them to the toolkit
Compute tangent vectors etc.
and add them to the geometry

```
int numPasses = myEffect.Begin();
for(int i = 0; i < numPasses; i++){
  myEffect.BeginPass(i);
  myMesh.DrawSubset(0);
  myEffect.EndPass();
}
myEffect.End();
```

Outlook 2

- Advanced programming features: branching, not unrollable loops
- Conflict with parallel processing, one has to pay in terms of performance.
- Nonetheless: Trend to overcome more restrictions in each new GPU generation.

Outlook 3

Future work?

- Try to put any algorithm onto the most current GPU?
- Conceive new ways to improve workflow in game and VR design