From Squish thru Zoom3 to ...

(a 64k intro experience)

Dmitry “AND” Andreev
and@intercon.ru

© Dmitry “AND” Andreev 2004

Overview...

- **64k intro background**
- **Sound subsystem**
  - History of the tools and synths
  - Basics and technical overview
- **Graphical subsystem**
  - 3D Scene creation
  - Scene storage in Squish and Zoom3
  - Shader stuff
- **Additional tricks**
  - Code size optimization, Blobs tricks

© Dmitry "AND" Andreev 2004

Sound synthesis

- **Synthesis Background**
- **Beginning**
  - 2000 - MSG (Minimal Sample Generator)
- **Squish 64k**
  - 2001 - SeqGEN (Sequence Generator)
- **Zoom3 64k**
  - 2002/2003 - AND XSynth

© Dmitry “AND” Andreev 2004

Synthesis Background

- **Simple waveform**
  - Like Square, Sine, Saw, Noise
- **Filter pipeline**
  - FIR or IIR
- **Additional effects**
  - Delay, Flanger, Chorus, Distortion, etc
- **Envelope stuff**
  - To control of synthesis parameters
- **Post processing**

© Dmitry "AND" Andreev 2004

MSG (non realtime)

- **ATG like, but for Sound**
- **MSG Basics**
  - 10 buffers
  - Render something to buffer
  - Use this buffer as a source for next operations
  - WAVE sample as result
- **Very hard to tweak**
- **Used with sample based stuff**
  - Like FastTracker + MINIFMOD

© Dmitry "AND" Andreev 2004

SeqGEN (non realtime)

- **Internally looks like MSG**
- **Fixed synthesis pipeline**
  - Sum of 8 Oscillators
  - Overdrive/Distortion filter
  - lowpass filter (IIR like) with resonance
  - Flanger effect
  - Delay
- **Set of 8 envelopes to control stuff above**
- **Simple Sequencer**
  - 16 tick pattern with Attack and Decay

© Dmitry "AND" Andreev 2004

AND XSynth (realtime) #1

- **Complete sound system**
  - C static library
  - Set of basic generators (Machine)
  - BUZZ machine wrappers (as Plugins)
  - File convertor from BMX to AMX
  - Custom music player (DXSound based)

- **Very fast and spetialized machines**
  - A lot of machines
  - A lot of links (connections)

© Dmitry “AND” Andreev 2004

AND XSynth (realtime) #2

- **Works in Realtime**
  - Easy to get a music sounds different
  - Very fast precalculation time
  - Hard CPU usage

- **Buildin MS SAPI Wrapper**
  - Some tricks to get the sound needed
  - Two synthesis ways for Win2k and WinXP
  - SAPI 4.0 (Win2k) and (SAPI 5.1)
  - But still sounds different and hard to tweak

© Dmitry "AND" Andreev 2004

AND XSynth (realtime) #3

- **Common synthesis pipeline**
  - Generation (Source of the sound)
  - Processing (DSP stuff)
  - Routing (Combine all machines together)

- **XSynth/BUZZ machine**
  - Internal 32bit float buffer (Mono/Stereo)
  - Set of patterns
  - Set of connections

© Dmitry "AND" Andreev 2004

AND XSynth (realtime) #4

- **Used in Zoom3 64k intro**
  - Zoom3 track takes 15% on 2GHz CPU
  - 66 Machines
  - 88 Connections
  - 60k of uncompressed AMX
  - 5k of compressed AMX (8%)
  - 6k of the library code

© Dmitry “AND” Andreev 2004

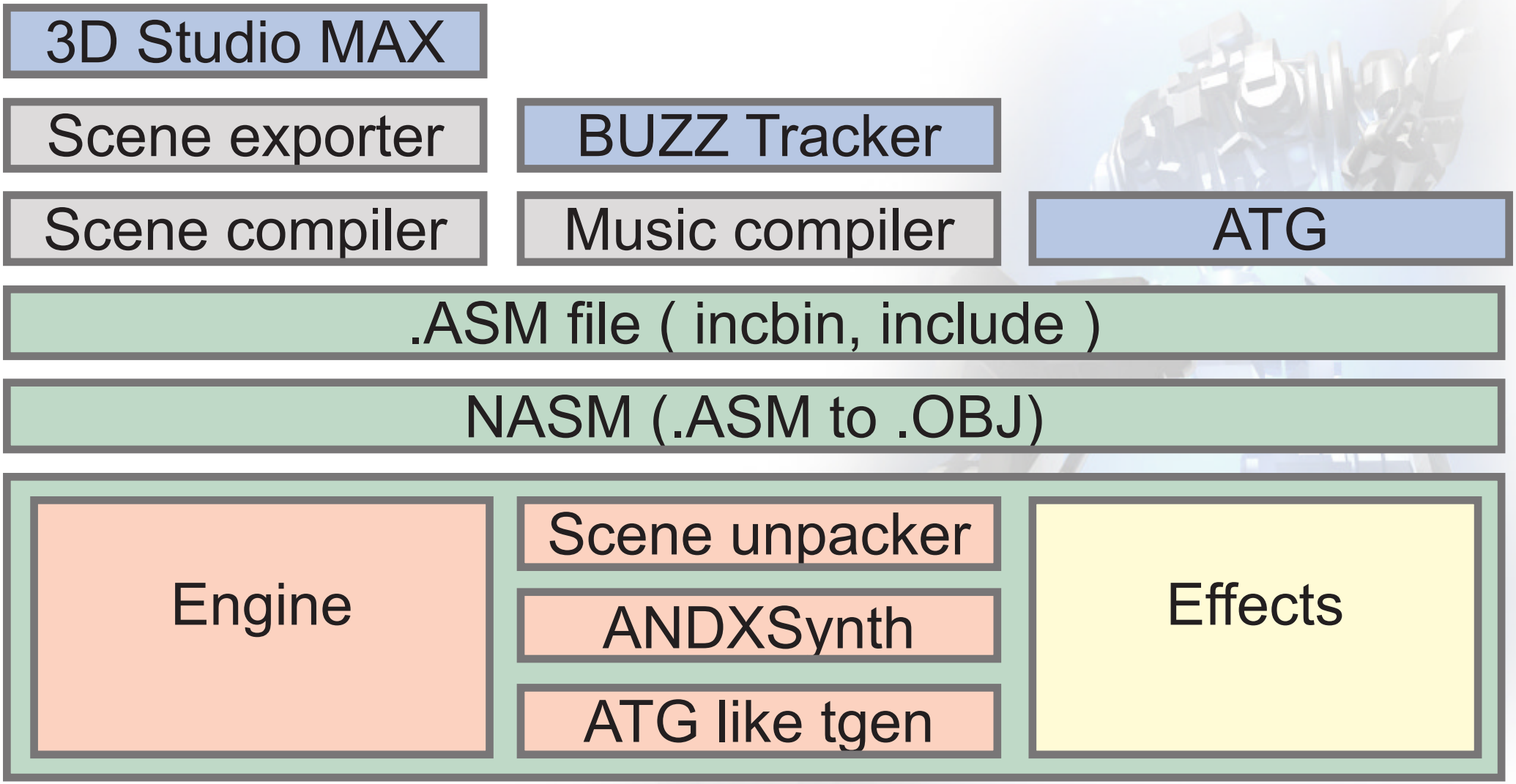
Squish/Zoom3 engine #1

- **Core interface**
  - Rendering API
  - Effects/Layers API
  - Timeline table
  - Scene stuff (Reconstruction, Rendering, ...)

- **Hardcoded**
  - Object/Scene managment
  - Lights managment
  - Effects, Animations, Cameras, etc...

© Dmitry “AND” Andreev 2004

Zoom3 building



© Dmitry “AND” Andreev 2004

Squish/Zoom3 engine #2

- **C like coded in C++**
  - Without classes (in most cases)
  - __fastcall functions

```
class CEffect_monsterx : public CLayer
{
private :
    .............
public :
    void        Init        ();
    bool        DoFrame     ();
    void        Release     ();
}
```

© Dmitry "AND" Andreev 2004

Zoom3 common code

```
bool CEffect_monsterx::DoFrame()
{
    if ( !CLayer::DoFrame() ) return false;
        rend_SetRenderFX( 0xffffff );
        CScene::Clear();
        PlayCamera( &g_pCamera01, 0.3f * m_fLocalTime );
        rend_SetViewProj( gc.Tgcs.g_xvCameraPos, .....
        XVec3 lpos1 = { -100, 40, 100 * sin( m_fLocalTime ) };
        static XVec3 ldif1 = { 0.7f, 0.9f, 1 };
        CScene::AddLight( lpos1, ldif1, XVec3_One, 300 );
        CScene::AddScene( m_pScene, lpos1, XEuler_Zero );
        rend_RestoreRenderFX( 7, 0 );
    return true;
}
```

© Dmitry “AND” Andreev 2004

From C++ to C like C++ #1

- **Math library used in Squish and Zoom3**
  - Almost 2 times smaller in C like code
- **C++ with classes**

```
class XVec3 {
public :
    float           x, y, z;

    friend XVec3 operator + ( const XVec3 &v1, const XVec3 &v2);
}
XVec3 operator + ( const XVec3 &v1, const XVec3 &v2 )
{
    return XVec3( v1.x + v2.x, v1.y + v2.y, v1.z + v2.z );
}
```

© Dmitry "AND" Andreev 2004

From C++ to C like C++ #2

- **C++ without classes**

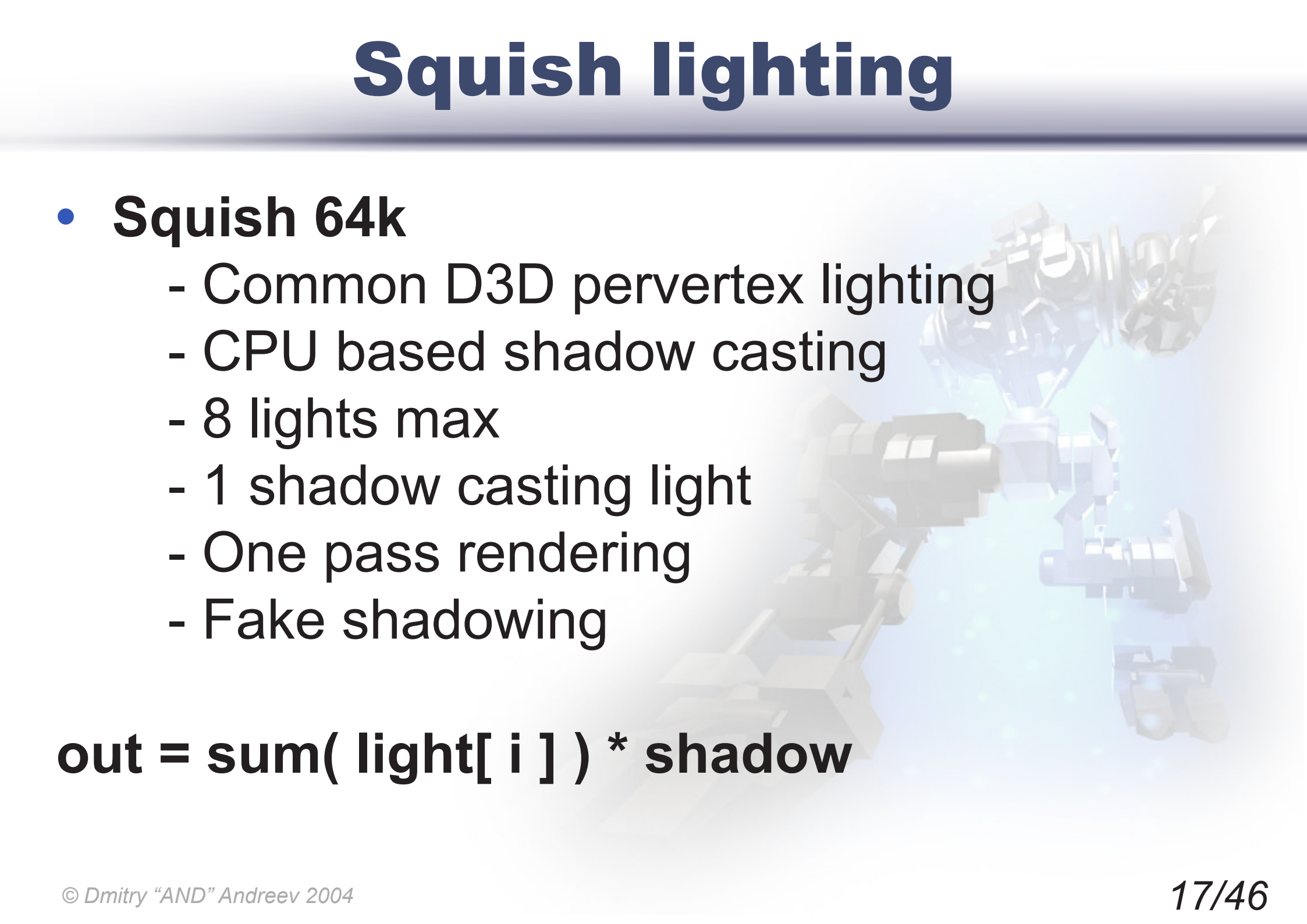
```
typedef float XVec3[ 3 ];

void __fastcall XVec3_Add
        (
        XVec3 xvOut,
        const XVec3 xvIn1,
        const XVec3 xvIn2
        )
{
        xvOut[ 0 ] = xvIn1[ 0 ] + xvIn2[ 0 ];
        xvOut[ 1 ] = xvIn1[ 1 ] + xvIn2[ 1 ];
        xvOut[ 2 ] = xvIn1[ 2 ] + xvIn2[ 2 ];
}
```

© Dmitry “AND” Andreev 2004

Squish lighting

- **Squish 64k**
  - Common D3D pervertex lighting
  - CPU based shadow casting
  - 8 lights max
  - 1 shadow casting light
  - One pass rendering
  - Fake shadowing

out = sum(light[i]) * shadow



© Dmitry “AND” Andreev 2004

Zoom3 lighting

- **The main feature (probably)**

- **Zoom3 Lighting is**
  - Perpixel model (PS1.1)
  - GPU based shadow casting
  - Fillrate limited number of lights (unlim.)
  - Low poly geometry looks fine
  - Multipass rendering
  - Complex Scene design ->

© Dmitry “AND” Andreev 2004

Zoom3 :: Scene design

- **A GAME of LIGHT and SHADOW**
- **Perpixel lighting looks better without complex diffuse textures**
- **More objects = more detail :)**
- **More shadows = more detail !**
  - Local light sources
  - More corners = more shadow casters
- **Abstract micro textures**
  - To hide pixelshader artifacts

© Dmitry “AND” Andreev 2004

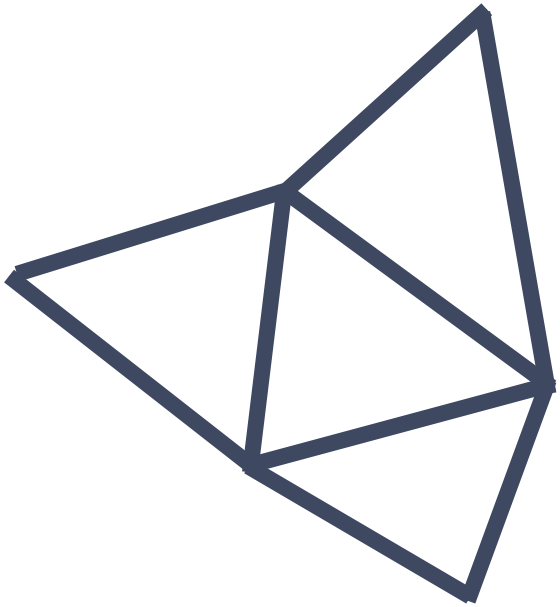
Zoom3 :: Lighing pipeline

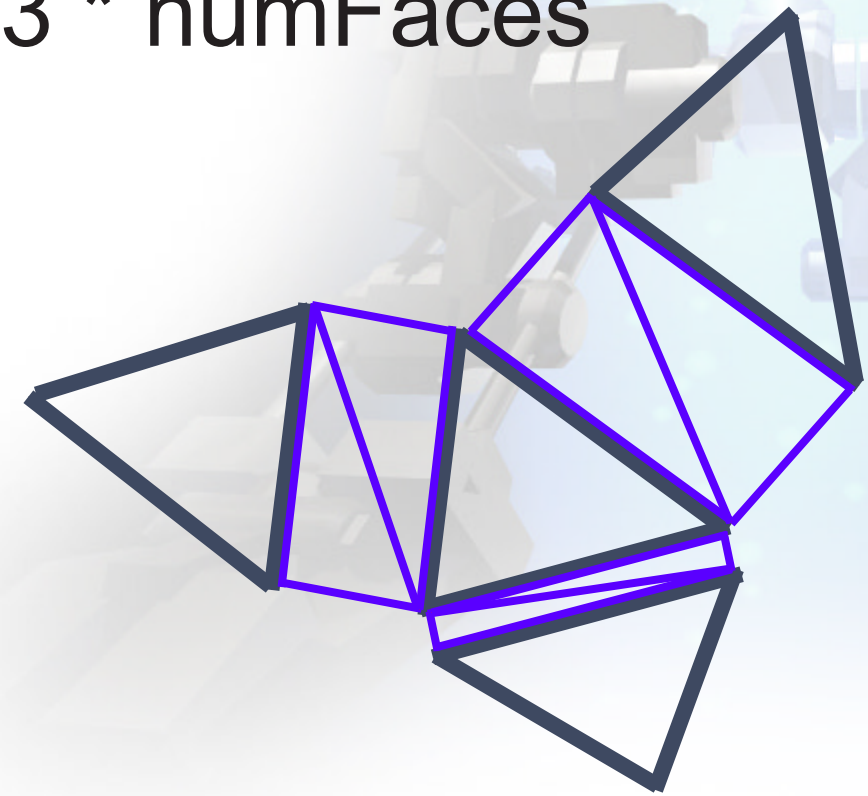
- **Render all objects to Z-Buffer**
  - Add Z based fog
  - Add Environmental mapping
- **For each light do**
  - **Compute mask of ushadowed area**
    - Common Stencil technique
  - **Render light to the frame buffer with Additive blending**
- **Final fullscreen post-processing**

© Dmitry “AND” Andreev 2004

Zoom3 :: Shadow casting #1

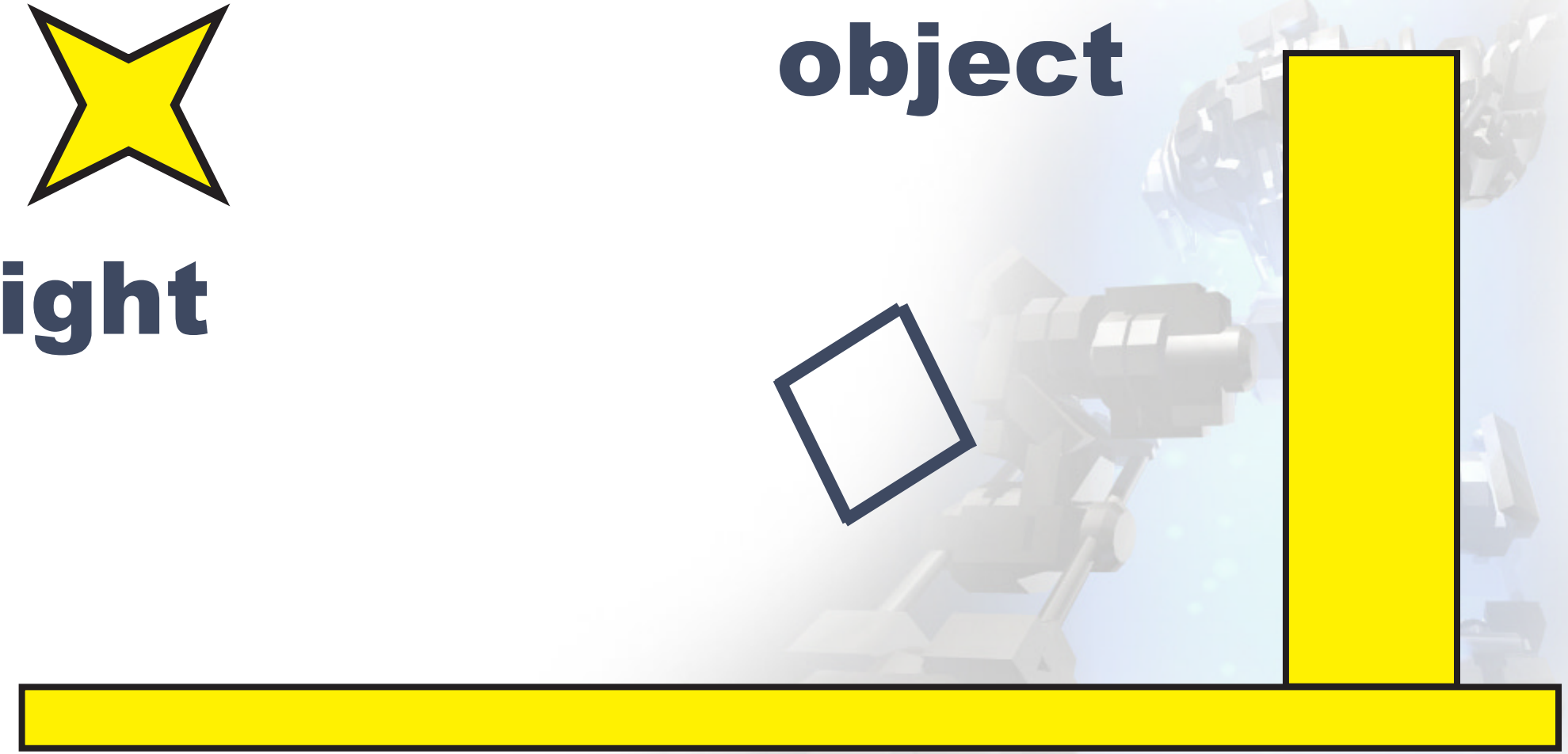
- **Vertex shader based shadow volumes**
  - Completely in GPU
  - 2 additional triangles for each original
  - numVertices = 3 * numFaces





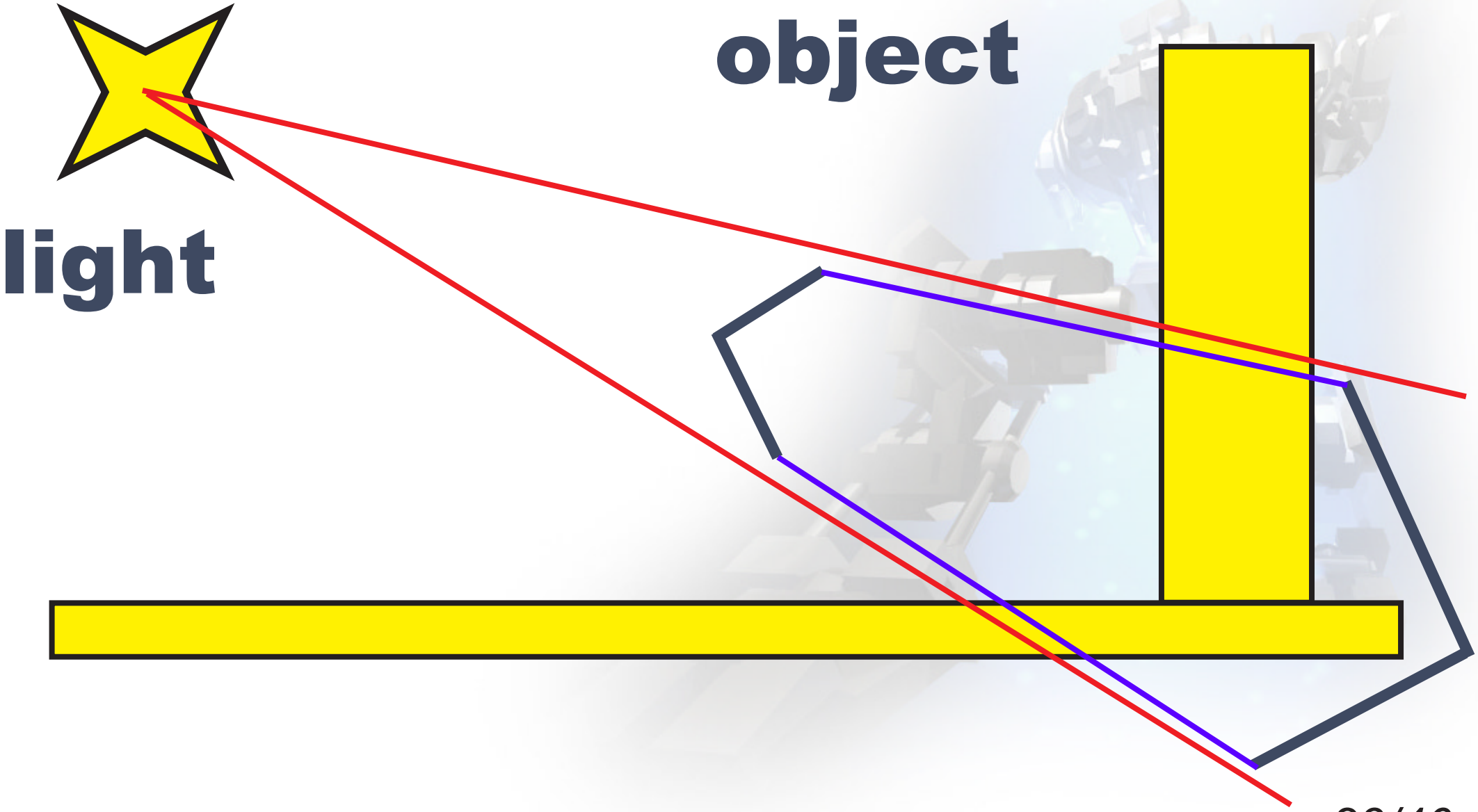
© Dmitry “AND” Andreev 2004

Zoom3 :: Shadow casting #2



© Dmitry “AND” Andreev 2004

Zoom3 :: Shadow casting #3



© Dmitry “AND” Andreev 2004

Zoom3 :: Shadow casting #4

- **Each vertex of the object has**

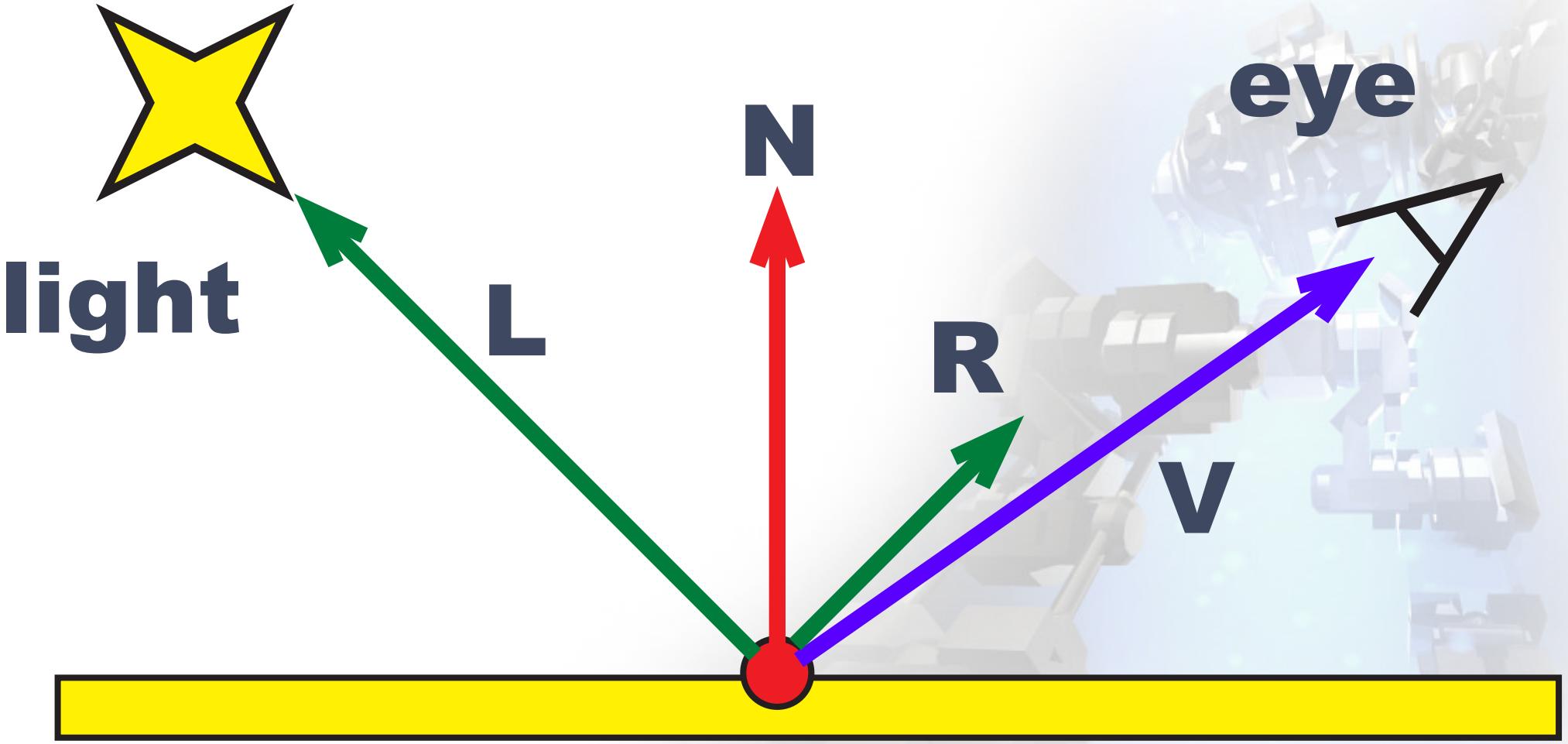
  - XYZ position

  - XYZ lighting normal - used for lighting
    (different for each vertex)

  - XYZ face normal - used for shaodw casting
    (equal for all vertices of the face)

© Dmitry “AND” Andreev 2004

Zoom3 :: Phong basics #1



© Dmitry “AND” Andreev 2004

Zoom3 :: Phong basics #2

- **All vectors are normalized**
- **Diffuse component**
  dOut = dot(L, N) * lightDiffuse
- **Specular component**
  sOut = pow(dot(V, R), f) * lightSpecular
  R = 2 * N * dot(N, L) - L

- **Final pixel is**
  Out = (dOut + sOut) * lightAttenuation

© Dmitry “AND” Andreev 2004

Zoom3 :: Pixelshader #1

ps.1.1

; c0 - Light diffuse
; c1 - Light specular
; v0 - “Normalized“ vertex normal

tex t0 ; Attenuation function (3D texture)
tex t1 ; Diffuse (micro) texture
tex t2 ; V - Normalized view vector (Cubemap)
tex t3 ; L - Normalzed light vector (Cubemap)

© Dmitry “AND” Andreev 2004

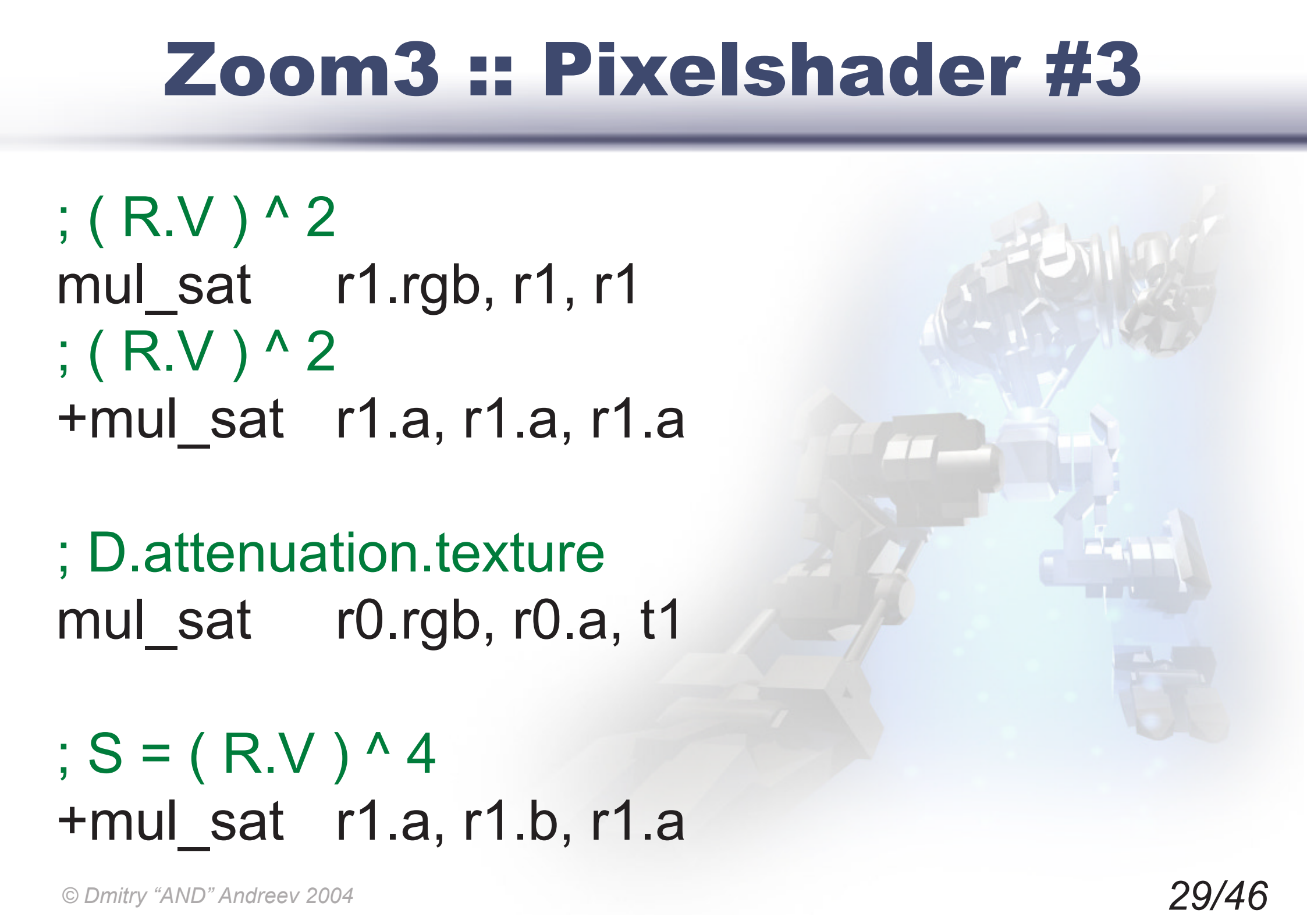
Zoom3 :: Pixelshader #2

```
; diffuse lighting D = ( N.L )
dp3             r0, v0_bx2, t3_bx2

; reflected light vector R = 2.N.( N.L ) - L
mad             r1.rgb, r0, v0_bx2, -t3_bias
; ( N.L ).attenuation
+mul_sat    r0.a, r0.a, t0.a

; Phong like specular S = ( R.V )
dp3_x2_sat r1, r1, t2_bx2
```

© Dmitry “AND” Andreev 2004

Zoom3 :: Pixelshader #3

```
; ( R.V ) ^ 2
mul_sat      r1.rgb, r1, r1
; ( R.V ) ^ 2
+mul_sat    r1.a, r1.a, r1.a

; D.attenuation.texture
mul_sat      r0.rgb, r0.a, t1

; S = ( R.V ) ^ 4
+mul_sat    r1.a, r1.b, r1.a
```



© Dmitry "AND" Andreev 2004

Zoom3 :: Pixelshader #4

```
; S.attenuation
mul_sat     r1, r1.a, t0.a
; S.attenuation.lightSpecular
mul_sat     r1, r1, c1

; out = D.atten.tex.color + S.atten.color
mad_sat     r0.rgb, r0, c0, r1
; kill a pixel if it is outside of the light (alphatest)
+mov_sat  r0.a, t0.a
```

© Dmitry “AND” Andreev 2004

Zoom3 :: Pixelshader #5

- **Extremely optimized pixelshader**
  - 12 instruction in PS.1.1
  - 8 instrucion slots used (max in PS.1.1)
  - 4 additional in alpha channel

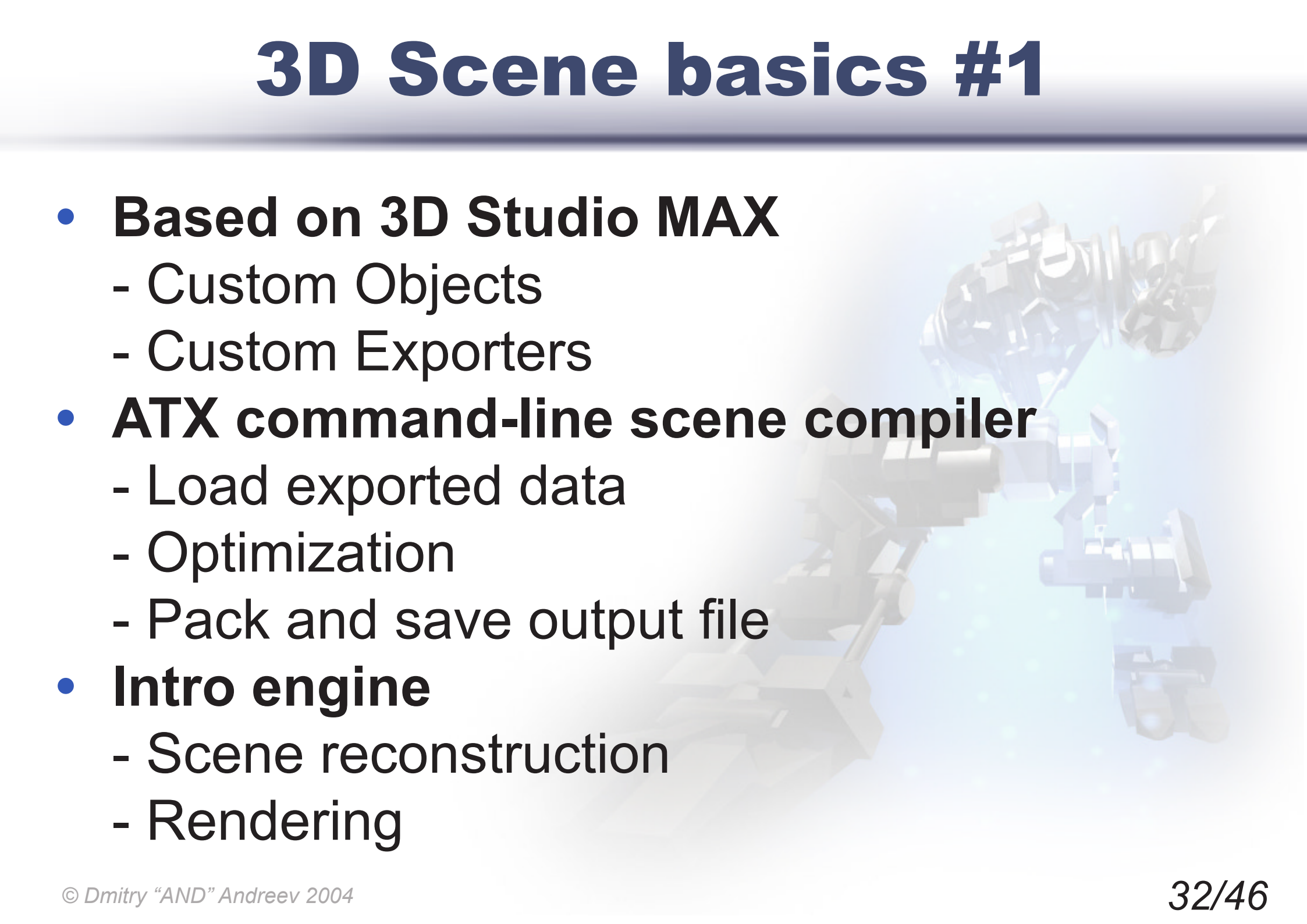
- **Final function is**

  out = ((N.L) * texture * lightDiffuse +
  ((V.(2.N.(N.L)) ^ 4) * lightSpecular)) *
  * lightAttenuation

© Dmitry "AND" Andreev 2004

3D Scene basics #1

- **Based on 3D Studio MAX**
  - Custom Objects
  - Custom Exporters
- **ATX command-line scene compiler**
  - Load exported data
  - Optimization
  - Pack and save output file
- **Intro engine**
  - Scene reconstruction
  - Rendering



© Dmitry “AND” Andreev 2004

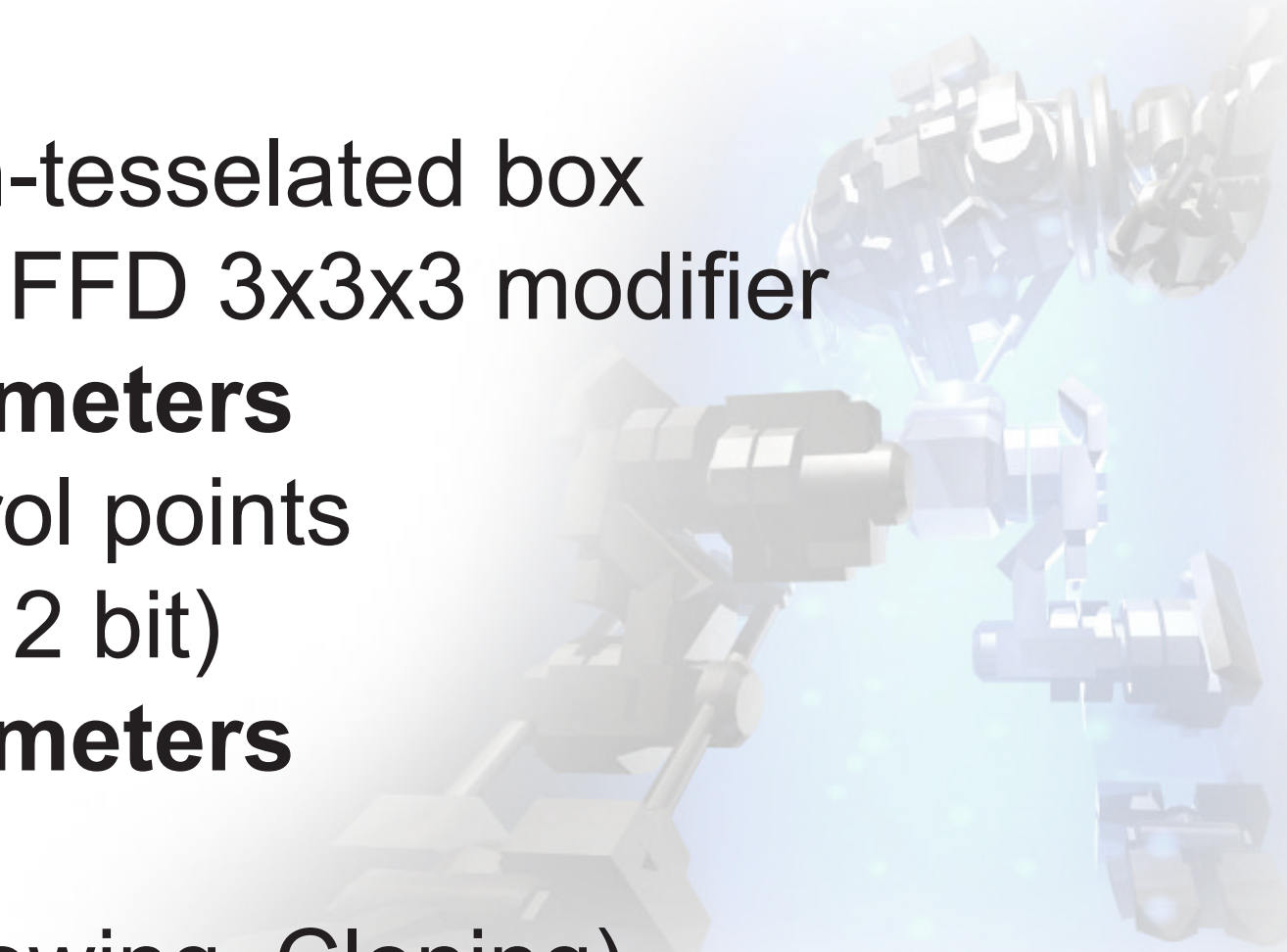
3D Scene basics #2

- **Simples objects**
  - Box, Cylinder, Sphere, etc...
- **Simple deformations**
  - Bend, Skew, Squeeze, Twist, etc...
- **General parameters of object**
  - Position, Rotation
- **Custom parameters of object**
  - Like width, height, radius, etc...
- **Cloned objects**
  - Position, Rotation, source of custom params

© Dmitry "AND" Andreev 2004

Squish :: General object

- **It is :**
  - Simples high-tesselated box
  - Plus applied FFD 3x3x3 modifier
- **Custom parameters**
  - 26 (27) control points
  - fake-floats (12 bit)
- **General parameters**
  - Texture ID
  - Flags (Shadowing, Cloning)
  - Delta position and rotation (for clones)



© Dmitry “AND” Andreev 2004

Value storage

- **Fake-floats (for Squish)**
  - Common float is 32bit
  - Fake-float is a part of common float

  xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
  xxxx xxxx xxxx 0000 0000 0000 0000 0000

  0x3f9e0610 = 1.23456f (32bit)
  0x3f900000 = 1.12500f (12bit)
- **Common integers (for Zoom3)**
  - 16bit short

© Dmitry "AND" Andreev 2004

Squish :: Scene storage #1

- **Number of objects**
- **Stream of object commands**
  - 8bit field

scxx xxxx

s = Is a shadow caster

c = Is a clone

x = Texture ID (64 textures)

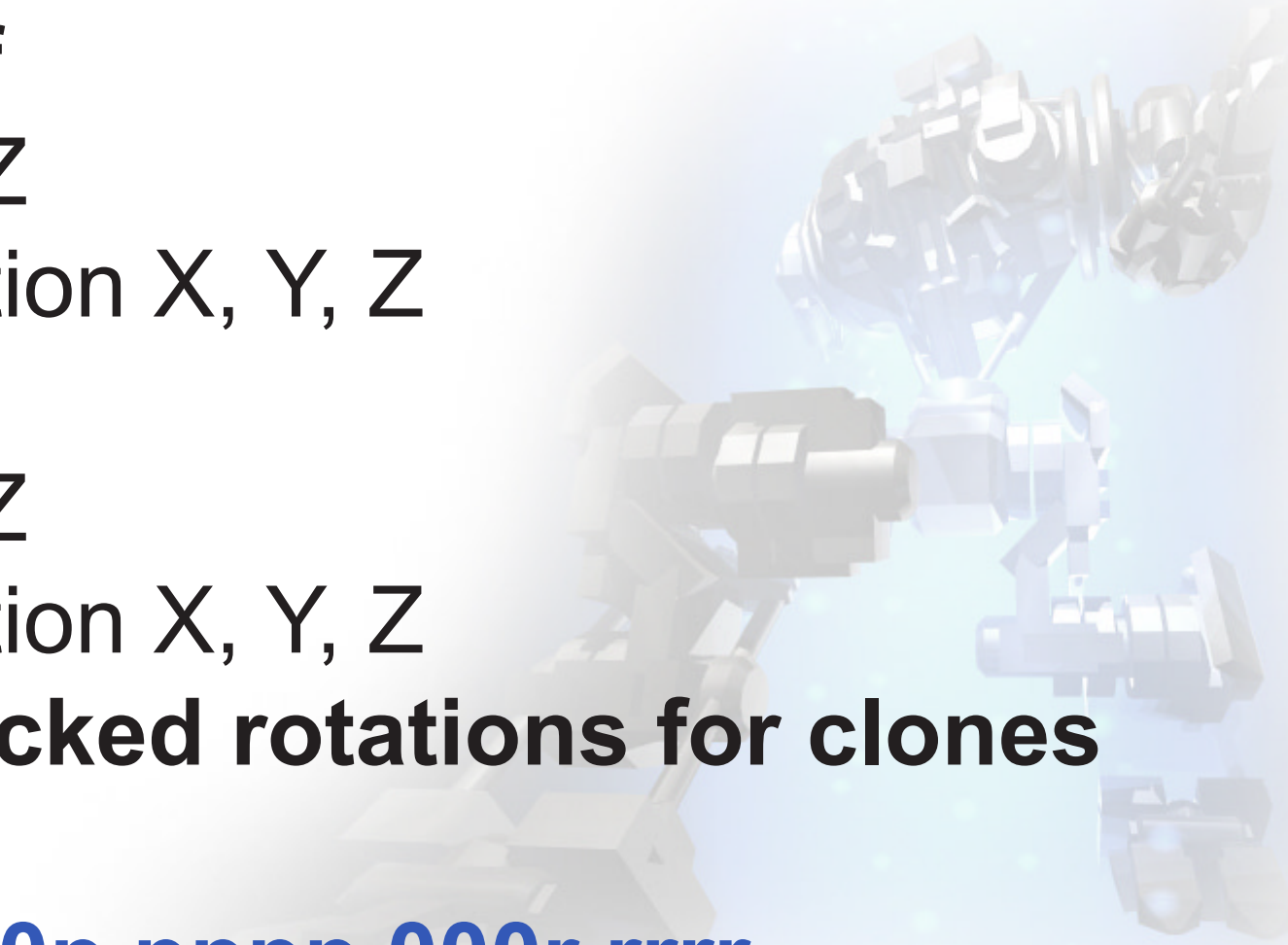
© Dmitry “AND” Andreev 2004

Squish :: Scene storage #2

- **High parts of**
  - FFD's X, Y, Z
  - Cloned position X, Y, Z
- **Low parts of**
  - FFD's X, Y, Z
  - Cloned position X, Y, Z
- **Stream of packed rotations for clones**
  - 24bit field

000y yyyy 000p pppp 000r rrrr

Yaw Pitch Roll



© Dmitry "AND" Andreev 2004

Zoom3 :: General object

- **It is :**
  - Cylinder based (mix of ChamferBox/Cyl)
  - Optimal for industrial scenes
- **Custom parameters**
  - Length (Rx), Width (Ry), Height
  - Fillet (Chamfer)
  - Number of height segments
  - Number of sides
  - Normal smoothing flag
  - Shadow casting flag

© Dmitry “AND” Andreev 2004

Zoom3 :: Scene storage #1

- **A3D signature**
- **Number of objects**
- **Stream of object commands**

  - 32bit field

  cccc cccc cccc cccc xxxx xmdd dsfq qqtt

  t = object type

  {

  q0 = SuperBox (has FFD)

  11 = Clone

  }

© Dmitry “AND” Andreev 2004

Zoom3 :: Scene storage #2

cccc cccc cccc cccc xxxx xmdd dsfq qqtt

x = Texture ID
d = Number of height segments
qqqt = Number of sides
f = Object has FFD modifier
s = No shadows
c = Number of clones

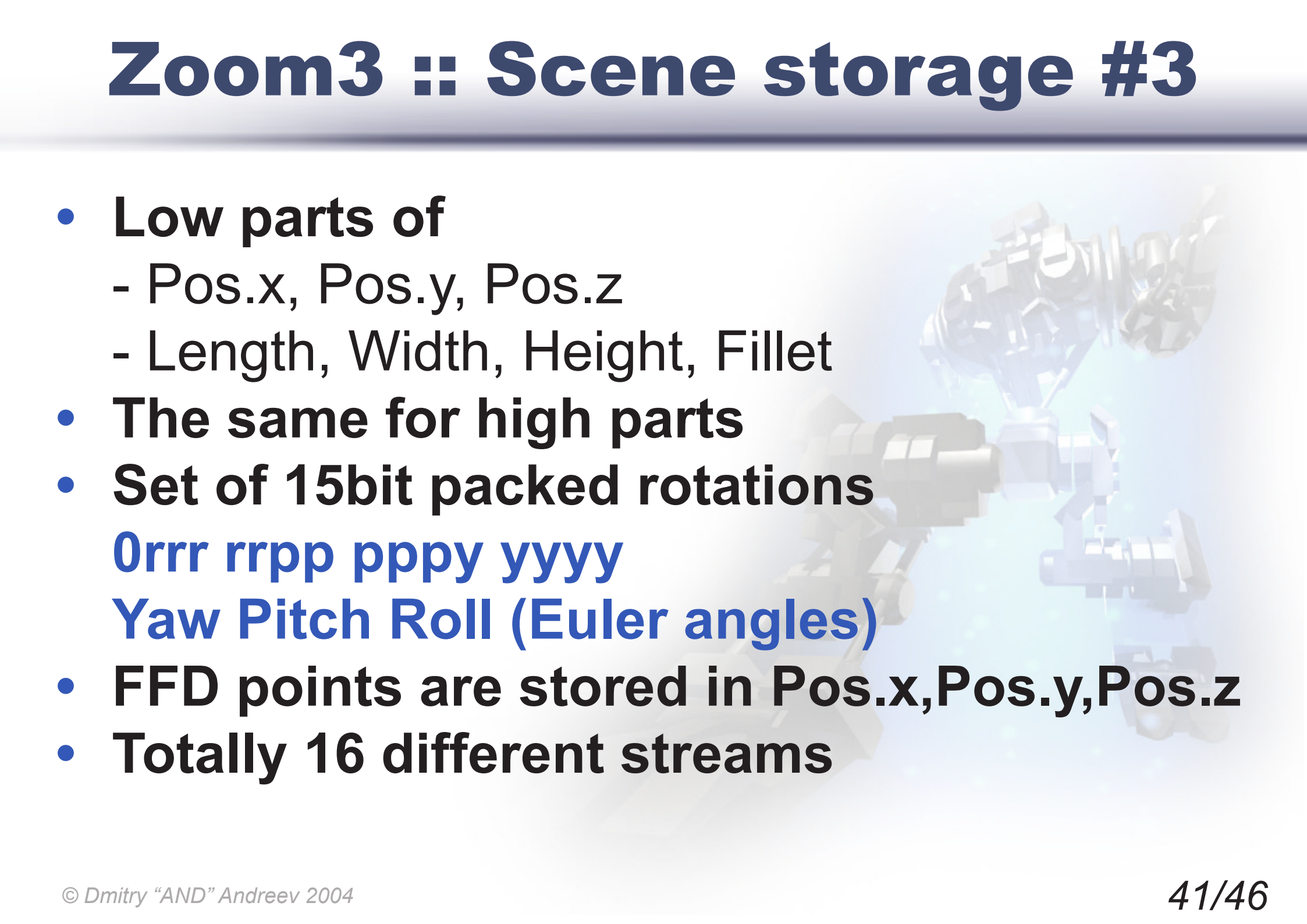
© Dmitry "AND" Andreev 2004

Zoom3 :: Scene storage #3

- **Low parts of**
  - Pos.x, Pos.y, Pos.z
  - Length, Width, Height, Fillet
- **The same for high parts**
- **Set of 15bit packed rotations**

  0rrr rrpp pppy yyyy

  Yaw Pitch Roll (Euler angles)
- **FFD points are stored in Pos.x,Pos.y,Pos.z**
- **Totally 16 different streams**



© Dmitry “AND” Andreev 2004

IsoSurfaces :: Basics

- **Originally is a scalar field**

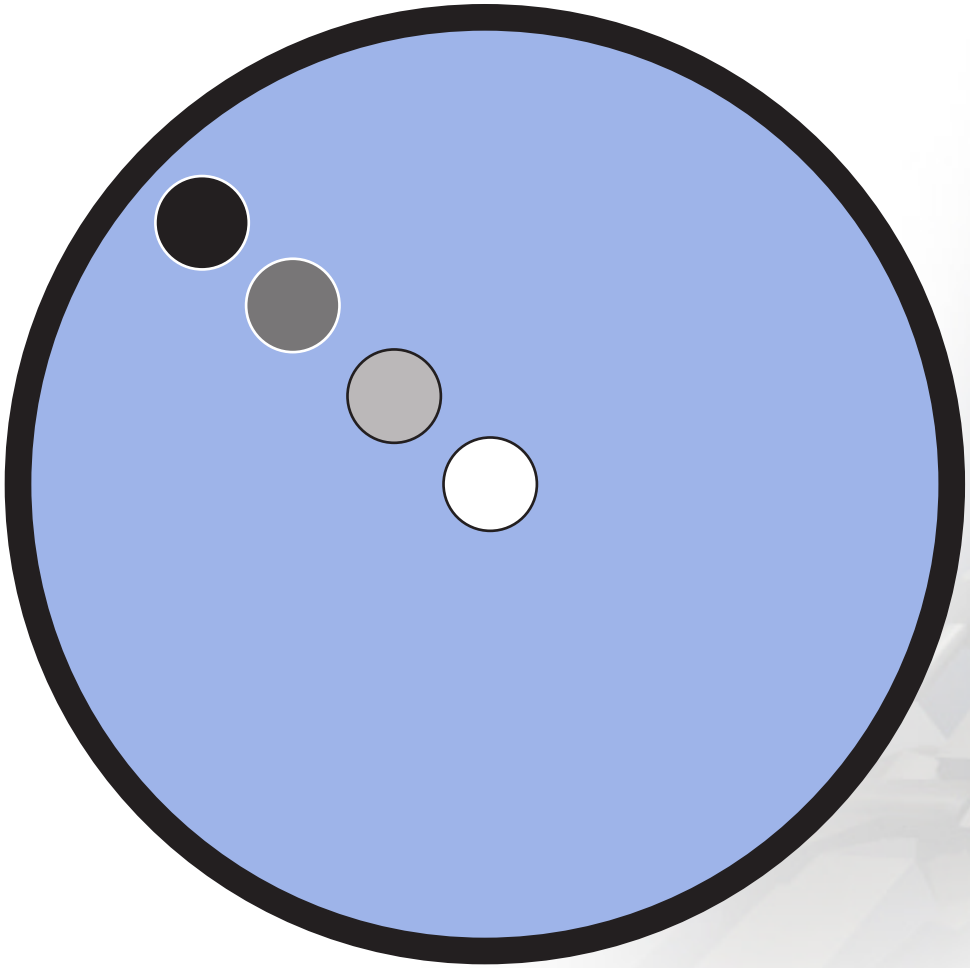
$$Fiso = \sum_{i=1}^{n} \left(\frac{Ri}{(x - Oix)^2 + (y - Oiy)^2 + (z - Oiz)^2} \right)$$

  - True in the case of Sphere-like surfaces
- **Complex surfaces**
  - Cube, Cylinder, Torus, etc...
  - Water worm (Like in Zoom3)
- **Hard to calculate surface and normals**

© Dmitry "AND" Andreev 2004

IsoSurfaces :: Scalar field

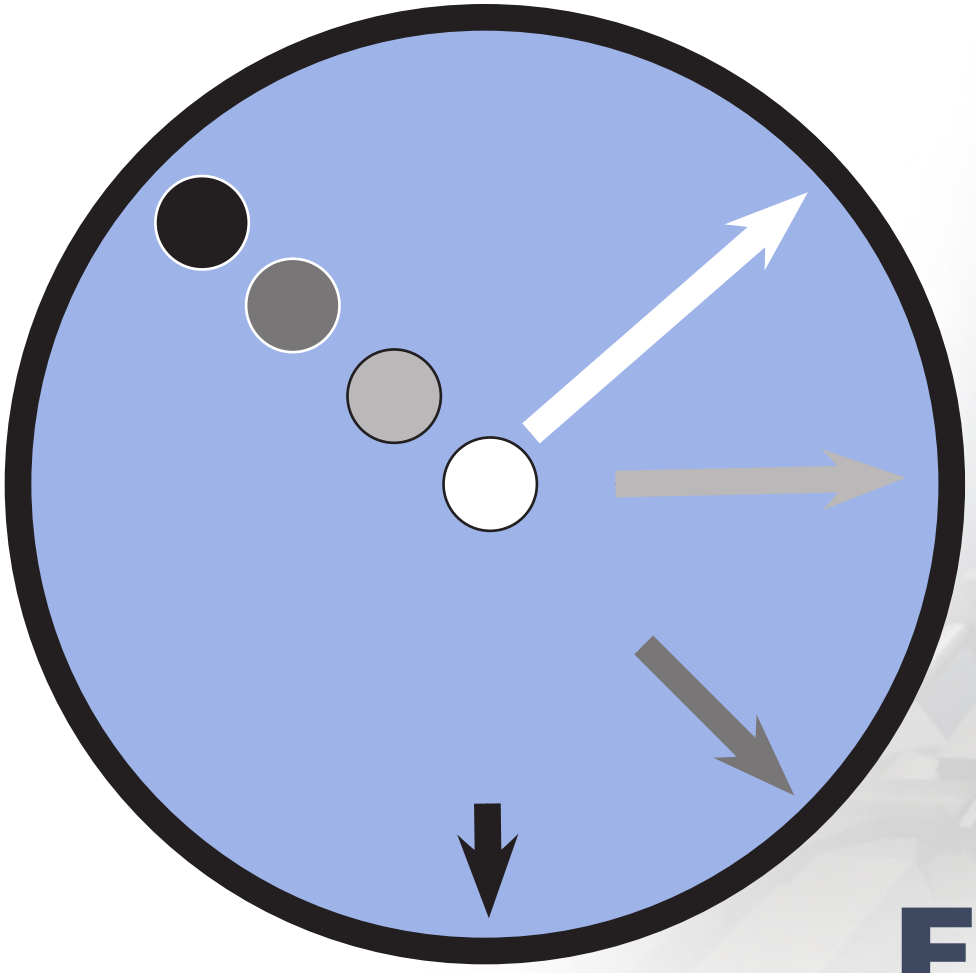
Scalar values



© Dmitry “AND” Andreev 2004

IsoSurfaces :: Combined field

Scalar values



Force field

© Dmitry “AND” Andreev 2004

IsoSurfaces :: 3D Sprites

- **Each iso-object is like a 3D sprite**
  - Precalculated 3D arrays

$$Fiso = \sum_{i=1}^{n} IsoObject[\,i\,].scalar$$

- **Normal is like a field direction**

$$Normal\_iso = \sum_{i=1}^{n} IsoObject[\,i\,].force$$

- **Works very fast and visually correct**

© Dmitry "AND" Andreev 2004

Questions

- **Email :**
  and@intercon.ru
- **Web :**
  http://and.intercon.ru

© Dmitry “AND” Andreev 2004