Making graphics in 4 kilobytes

Iñigo 'iq' Quilez / rgba

www.rgba.org/iq

10/05/2008 Errenteria

Index

Index

4k executable graphics

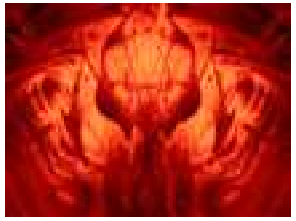
- Misconception: “4k graphics are not interesting, it’s just a 4k intro without animation and music”. Wrong
  - People expect (or so they should) to see more, better, and nicer images
  - Basically, the responsibility is bigger.
- A big variety of techniques possible
  - Whited raytracing, pathtracing, polygons plus shaders, just polygons, just shaders, cpu rendering, vector graphics, raymarching, voxels, splating, point clouds, fractals algorithms, photoshop strokes capturing, mesh subdivisions…
  - Choose your weapon!
- So, how to approach the challenge of drawing a beautiful image in 4 kilobytes ?

gfx

2D

vectorial

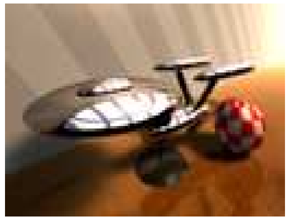
procedural



compressed

3D

vectorial or primitives



procedural

gfx.2D.vectorial

“headfunk“, by d-lab42, 4th at Breakpoint 2008

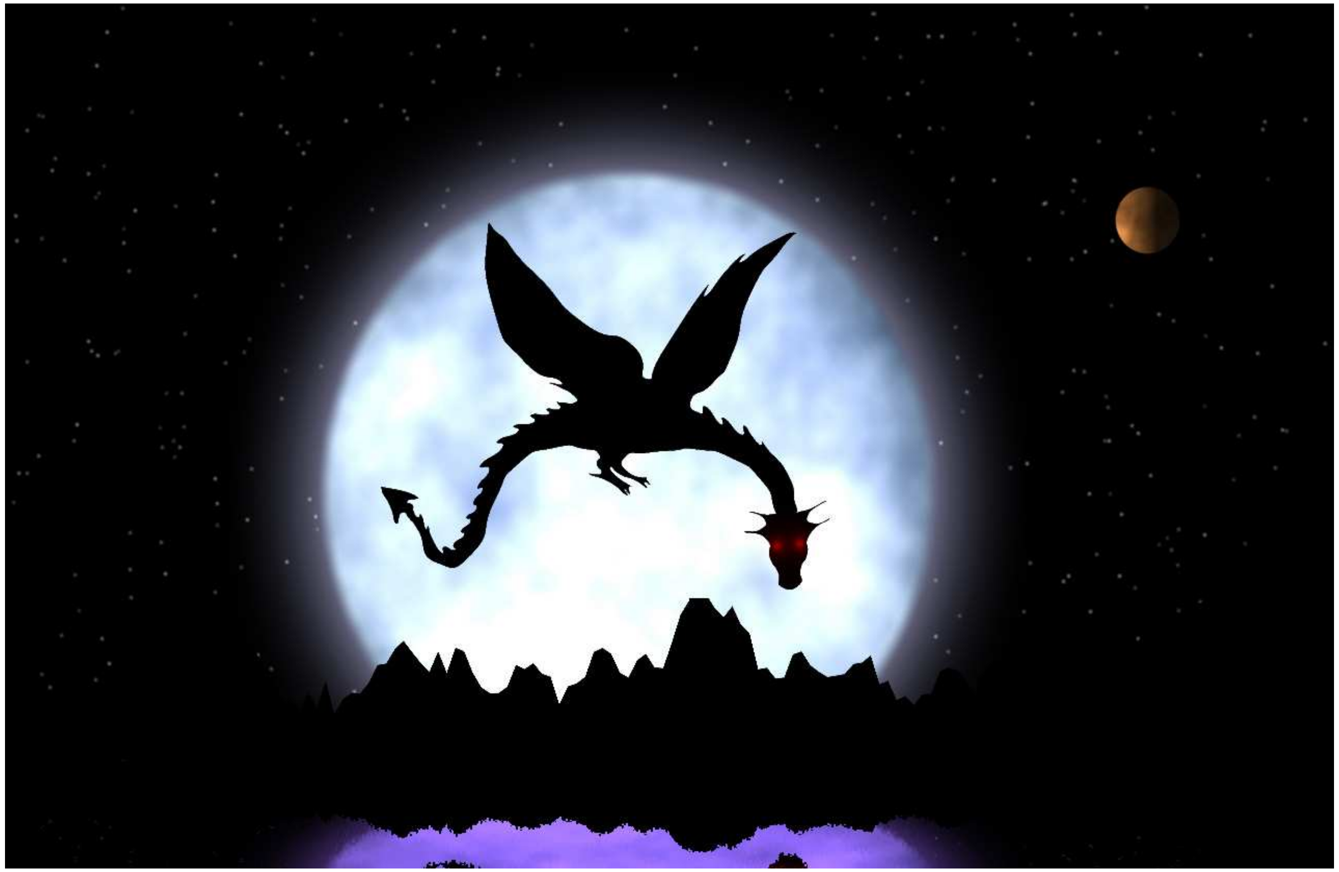
- Designed in InkScape
- Exported to SVG and converted to arrays of C data.
- Pixel based drawing: fill(x,y,color), circle(x,y,rad,color), etc

gfx.2D.vectorial

“3 Minutes“, by mercuri, 2nd at TUM 2006

- Cubic bezier segments.
- Pixel based drawing: fill(x,y,color), circle(x,y,rad,color), etc

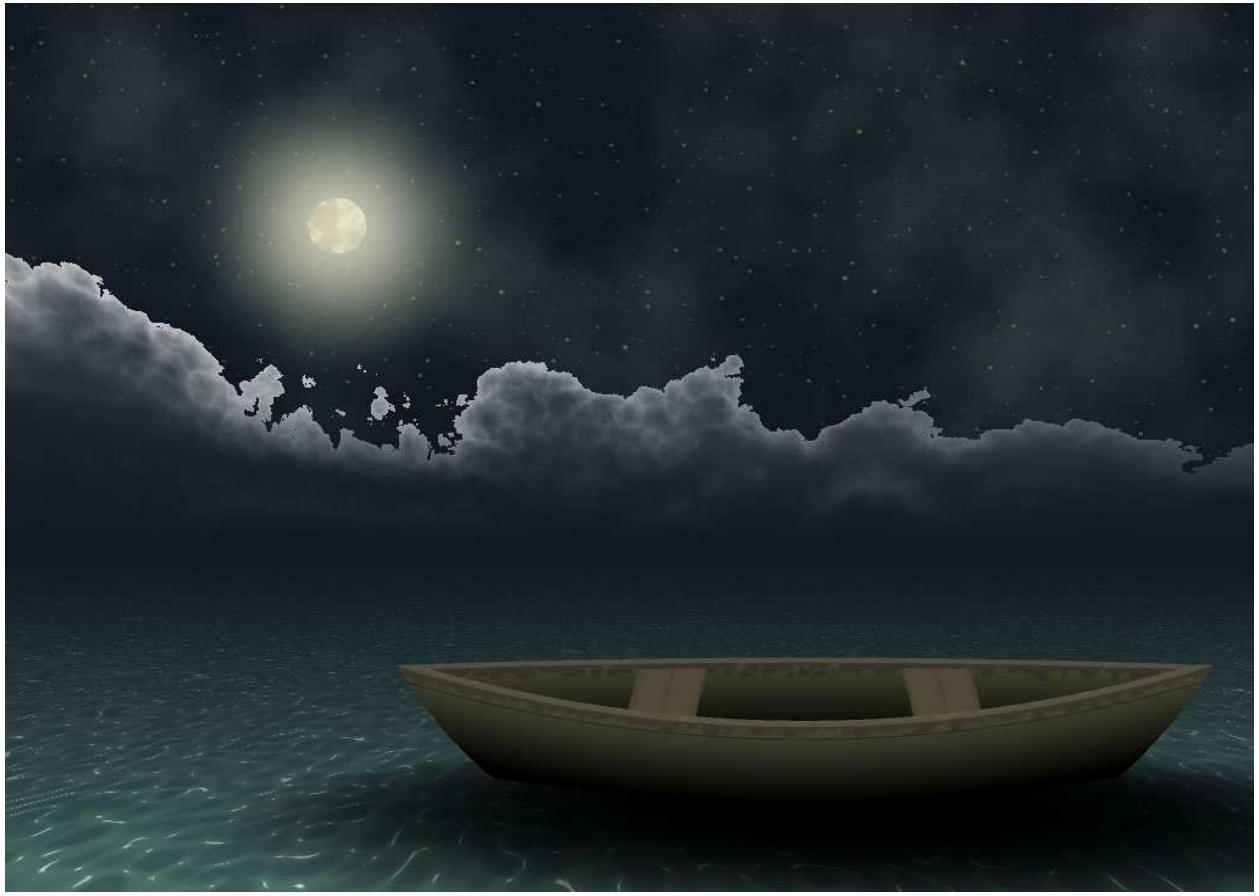
gfx.2D.vectorial



“Lunreal“, by mercuri, 6th at Breakpoint 2008

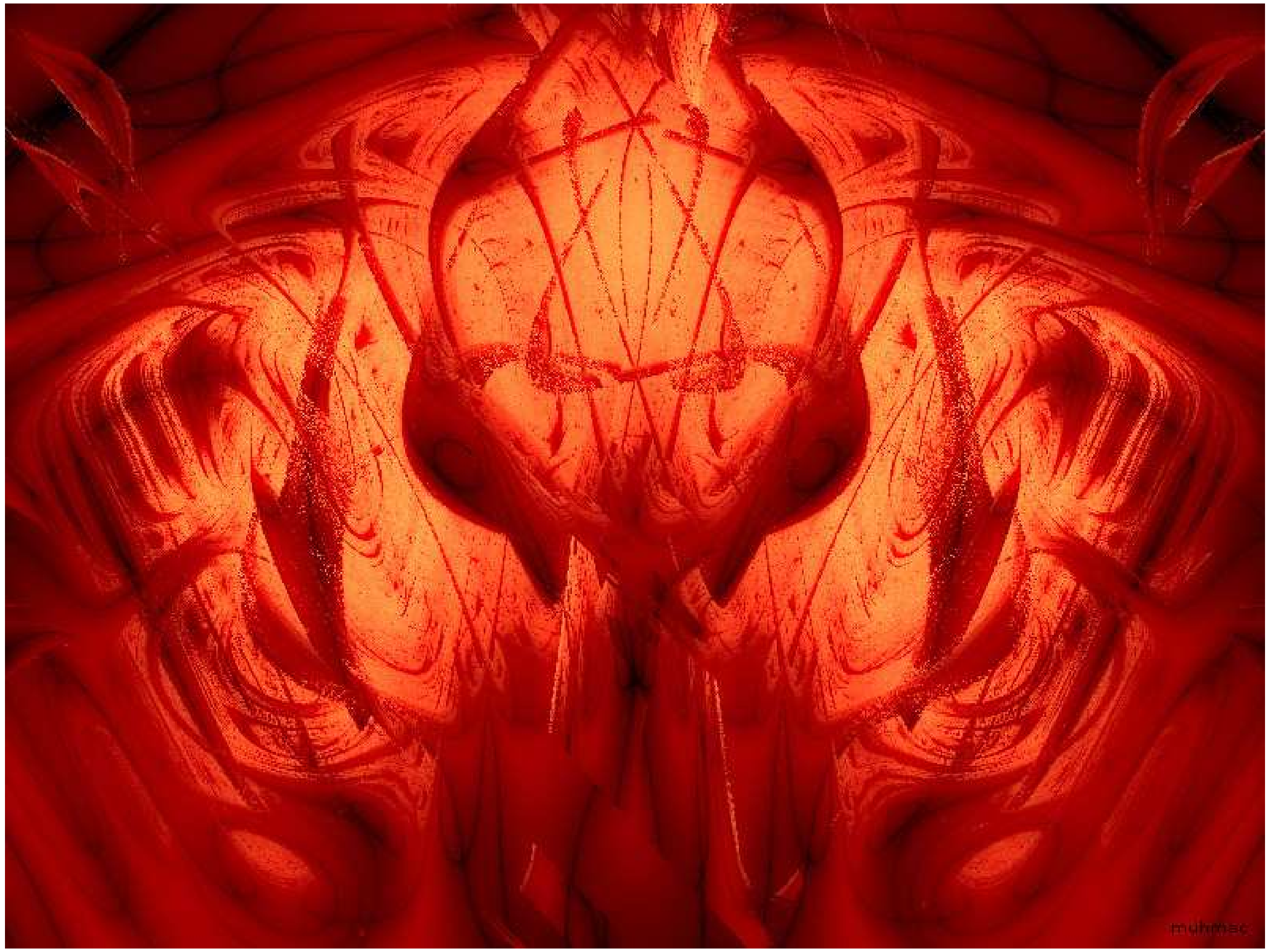
- Cubic bezier segments.
- Pixel based drawing: fill(x,y,color), circle(x,y,rad,color), etc
- Several mixed techniques
    - 3d sphere
    - buffer reflections
    - procedural texture

gfx.2D.procedural


“Lonely Boat“, by Digimind, 1st at TUM 2007

- Fully procedural image.
    - Extensive use of perlin noise
    - Coordinates based shadows

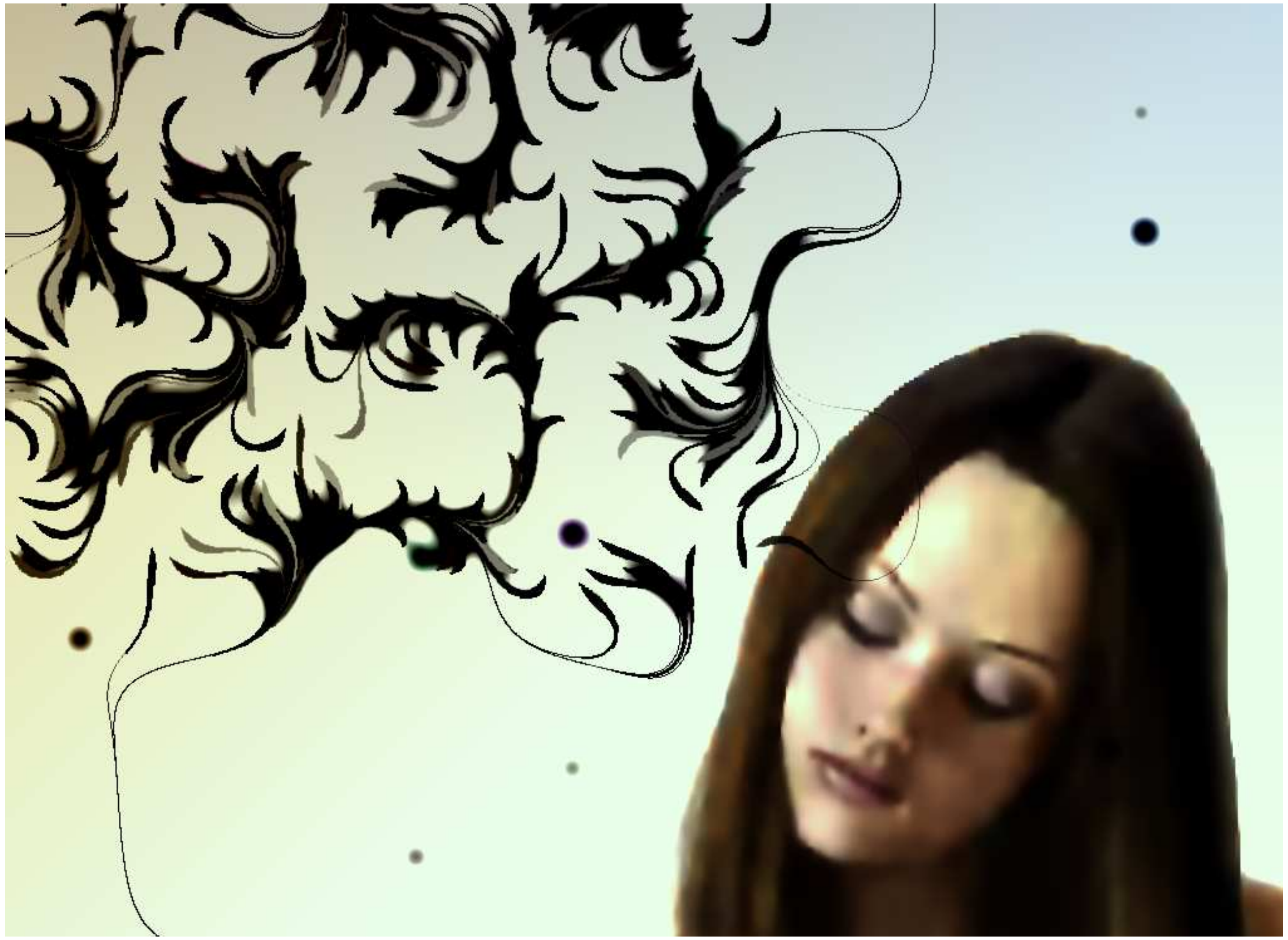
gfx.2D.procedural



"tiphareth", by Speckdrumm, 1st at TUM 2006

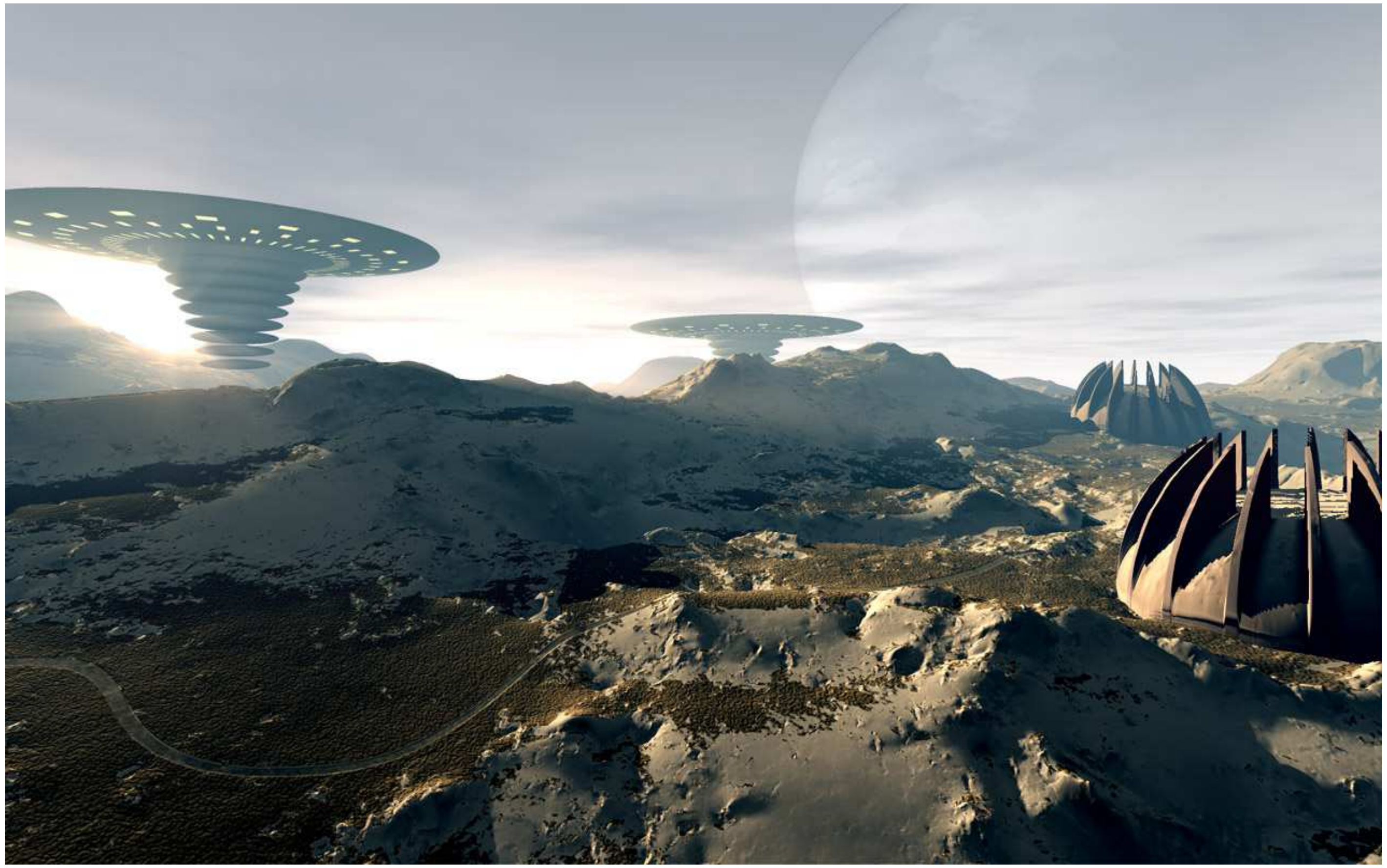
- Lyapunov fractal

gfx.2D.compressed


“inslexia“, by rgba, 4th at TUM 2007

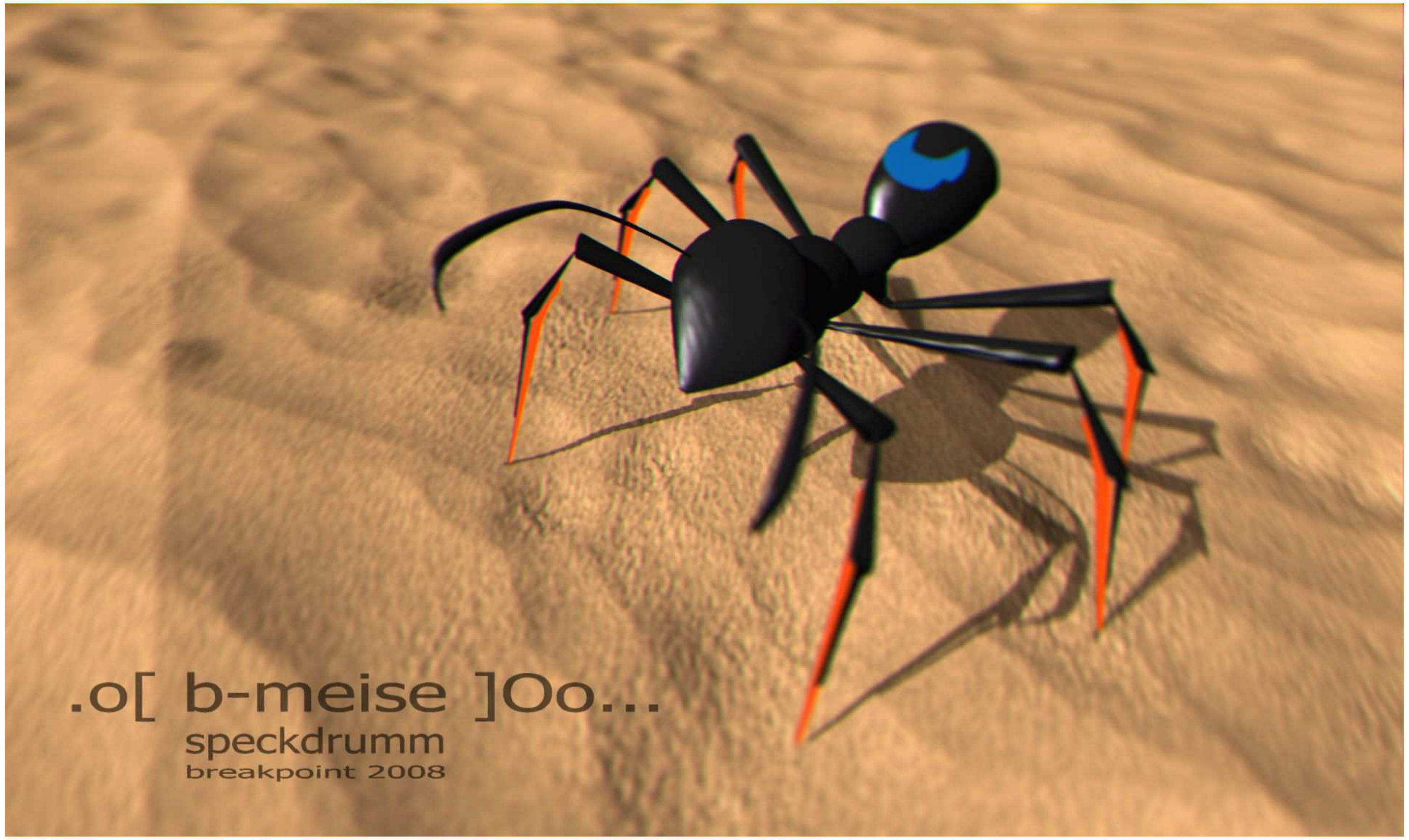
- Wavelet image compression
    - Haar basis
    - Varianze based blurring
    - YUV decomposition
- Procedural ornaments (perlin)

gfx.3D.procedural


“ixaleno“, by rgba, 1st at Breakpoint 2008

- Procedural lanscape
  - terrain
  - alien bases
  - road
  - texturing
- Raymarching and raytracing
- Other CG effects
  - light shafts
  - soft shadows
  - atmospheric scattering

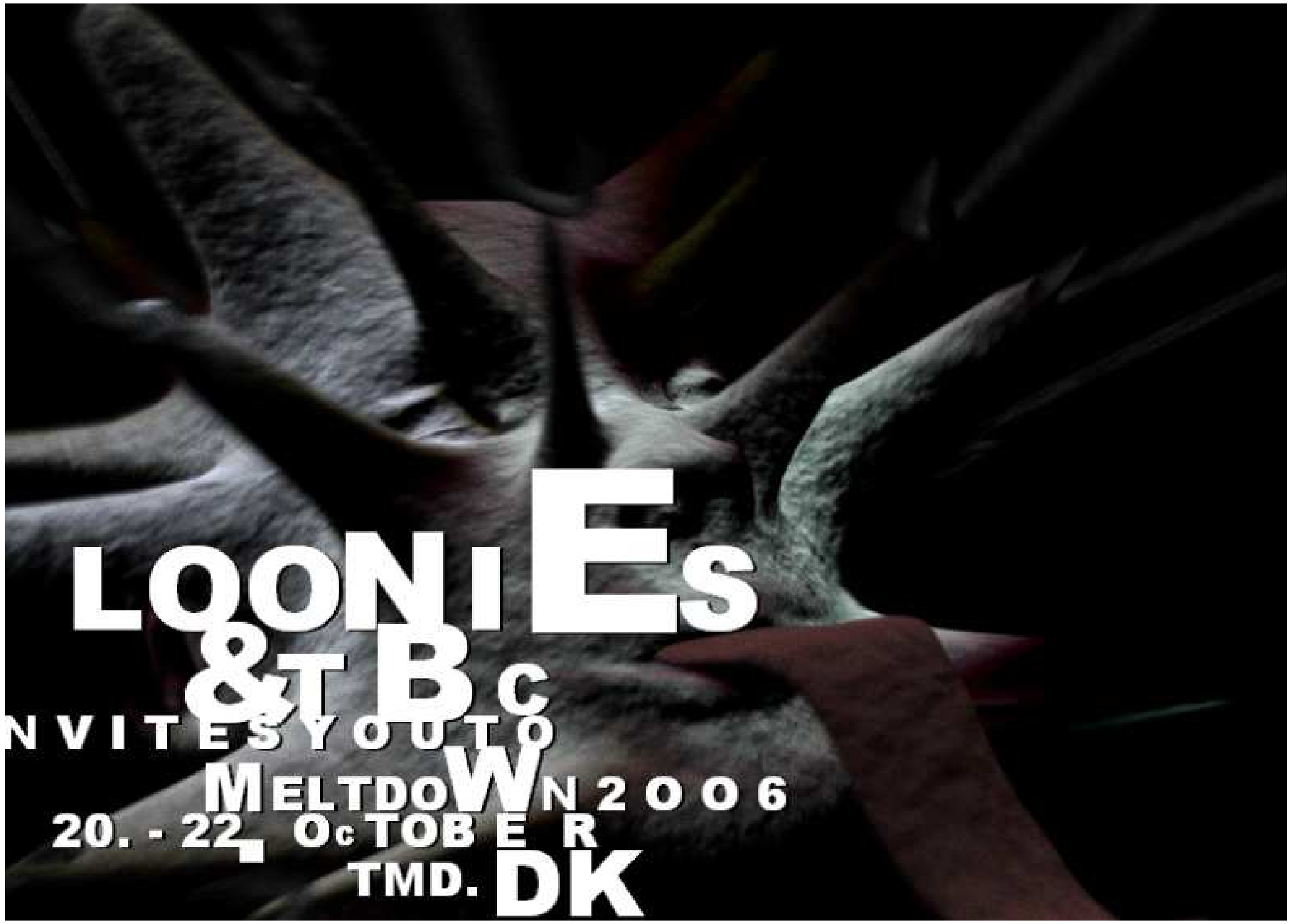
gfx.3D.vectorial



“headfunk“, by Speckdrumm, 3rd at Breakpoint 2008

- Polygonal mesh, rendered in OpenGL
- Exported from Cinema4D
- Several techniques from 3D CG:
    - Shadows
    - DOF
    - Color disortion
- Procedural terrain

gfx.3D.vectorial



“mtbshbw“, by Loonies & TBC, 4th at Buenzli 2006

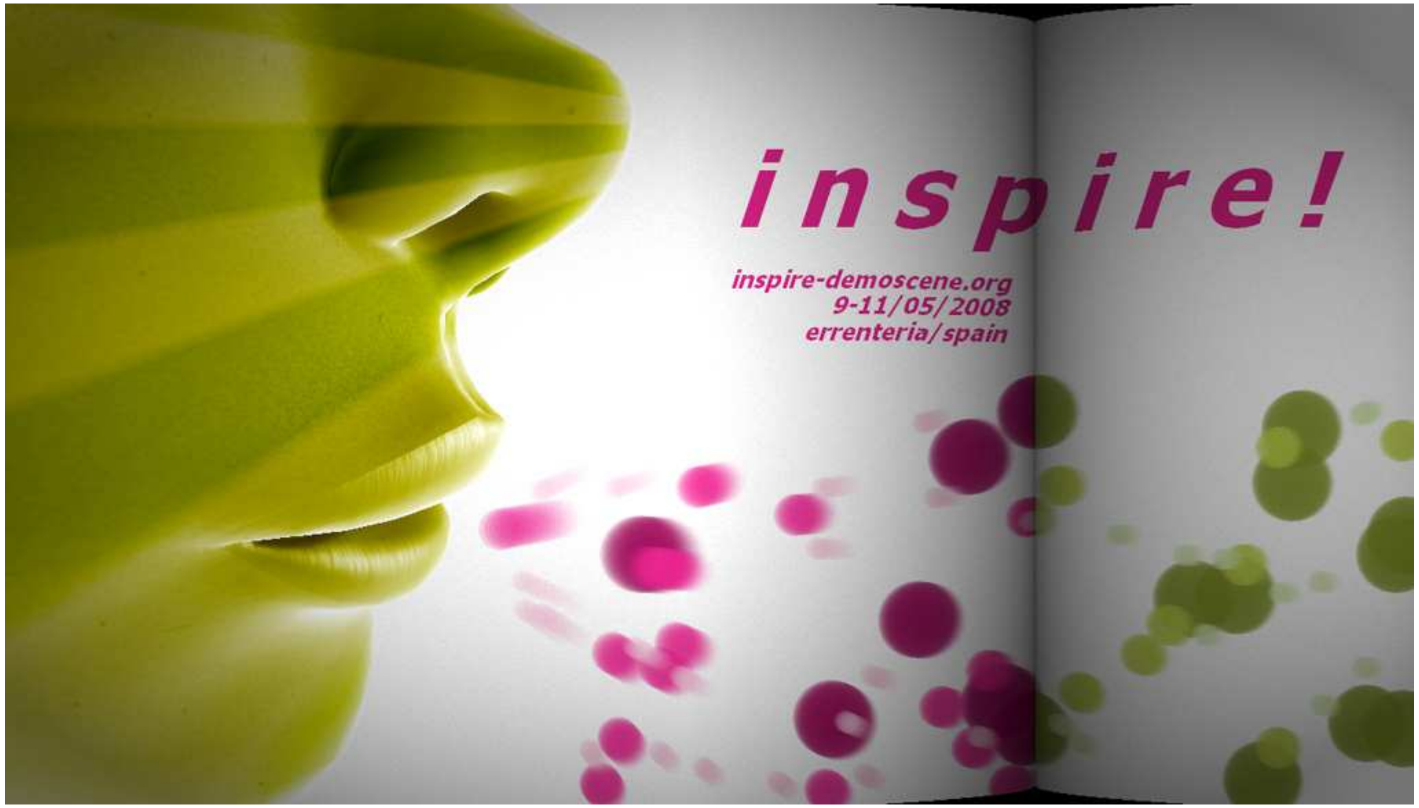
• Polygonal mesh, rendered in Direct3D

• Typography

• DOF

• Procedural texture

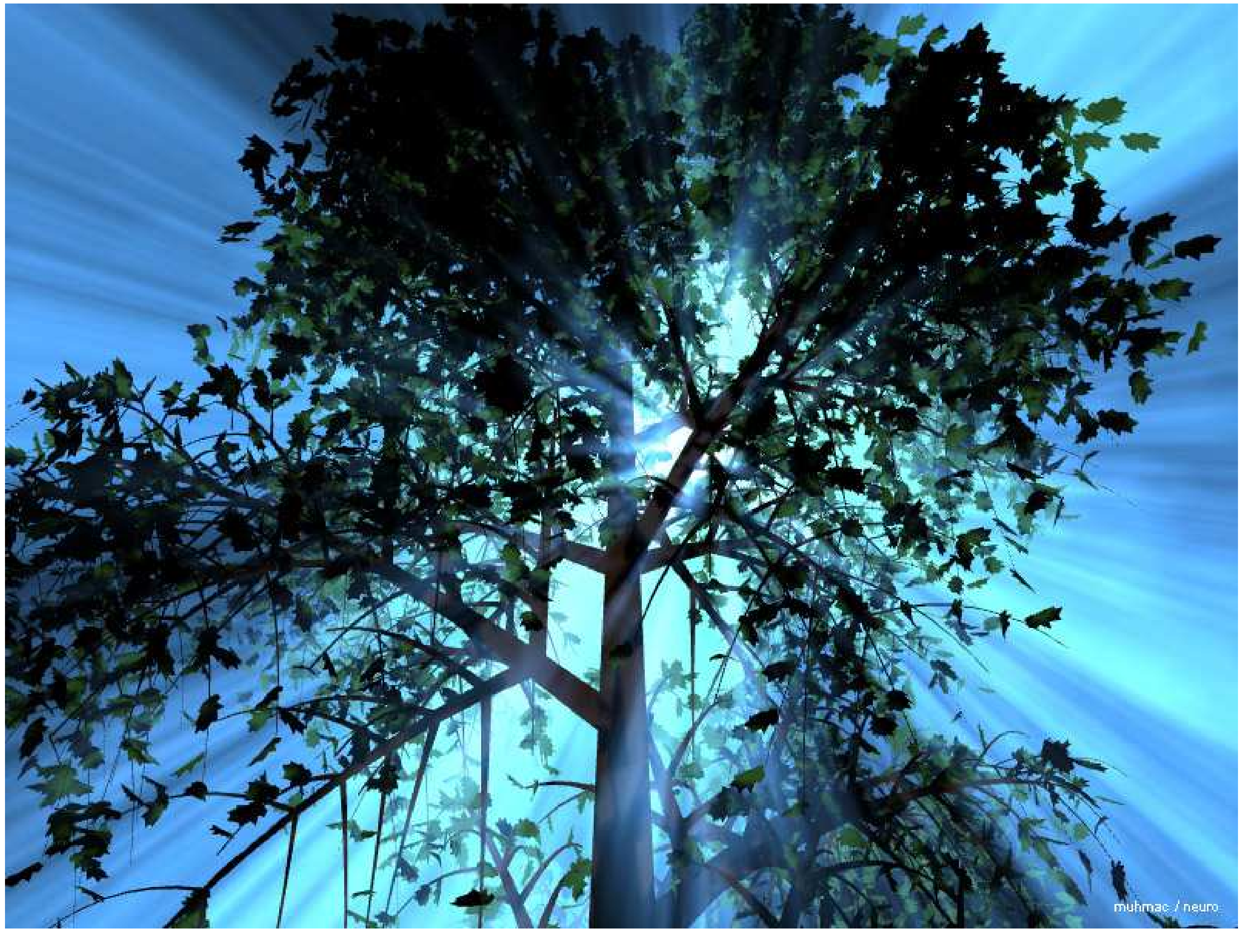
gfx.3D.vectorial



“elexiane“, by rgba, no competition

- Polygonal mesh
  - Compressed
  - Subdivided
- Raytraced on the CPU
- Exported from Maya
- Perlin noised skin shader
- Procedural motif

gfx.3D.vectorial



"der_wald_stirbt", by Neuro, 1st at Buenzli 2006

- Polygon renderer
- Direct 3D (shaders 2.0)
- Procedural placement of the primitives

gfx.3D.vectorial

“off the shelf“, by Loonies, 2nd at Breakpoint 2008

- Pathtracing
- Quadratic primitives + CSG
- All in GLSL (shaders 3.0)
- HDR

Index

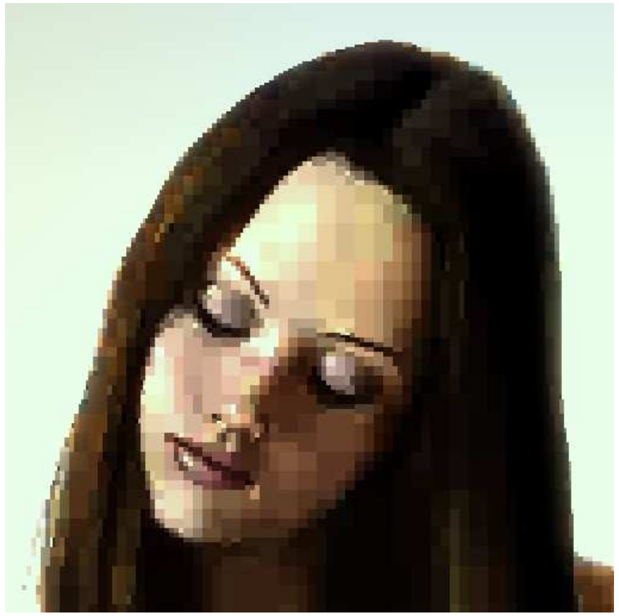
Image compression

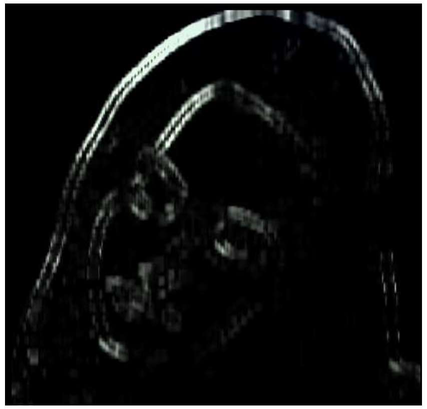
- Simple Haar wavelet encoding. Idea:
  - Recursively predict pixels based on lower resolution version of the image.
  - Reduce/increase resolution by two each time, store differences
  - Store differnce coeffients for each level and position (band), in a different array, in Morton order.
  - Incredibly tiny code for decompressor.

Image compression

• Problem, high energy coeficient removal introduces big errors. In the case of the Haar basis the errors have block shapes.

• Hack, detect the block and blur them (blur more on areas of big blocks, and keep image edges) : use a variance detector driven gaussian blur.


Decompressed image, lot of artifacts


Variance of the blocky image

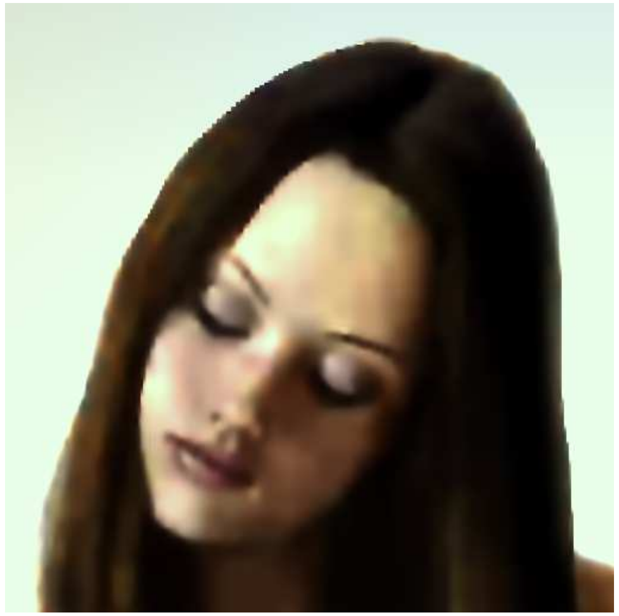

After adaptative bluring

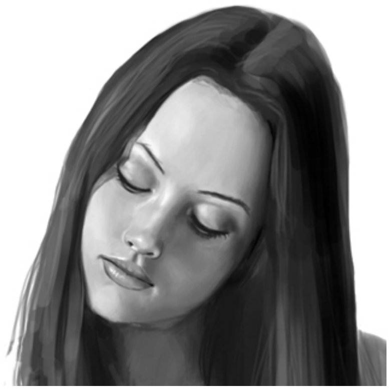
Image compression

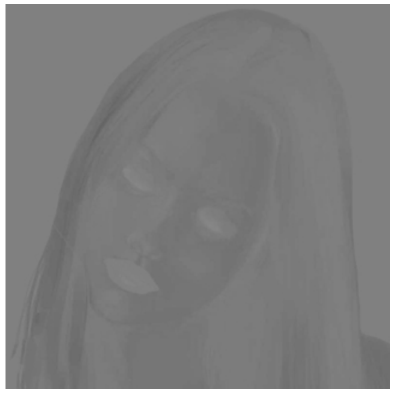
• Like in JPEG, do not encode RGB. First, decorrelate the three signals (remove common information) by converting to YUV. Y is gray level or luminance, UV are chroma (B-G) and (R-G).

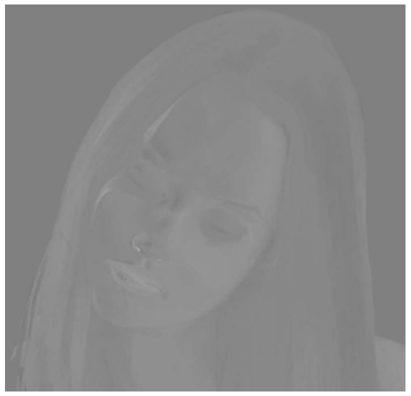
• Subsample UV by 4 in each dimension, nobody will notice.

• UV will be for free. In Inslexia, Y was 1300 bytes, U and V 80 bytes each (so basically compressing color images is as expensive as monochrome images).

RGB


Y

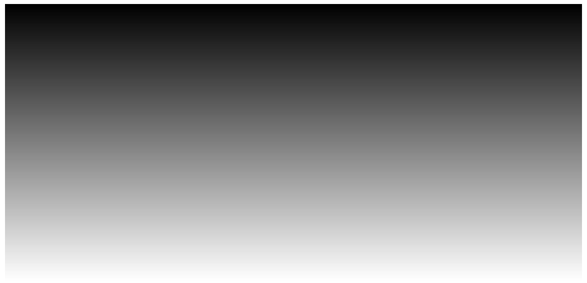

U


V

Index

2D Procedural drawing

- The idea is, given the coordinates of a fragment, output a color.
- Fragment coordinates go from 0 to 1,
    - so images are resolution independent.
    - so supersampling or image distorions can be done for free.

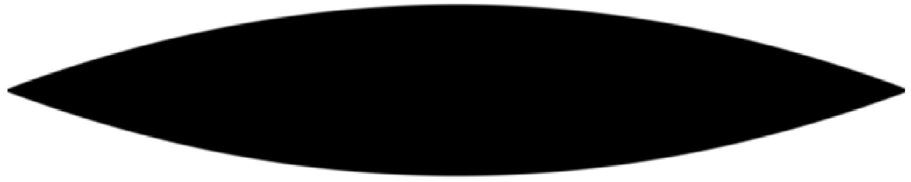


```
void draw( float *rgb, float x,
                       float y )
{
    rgb[0] = x;
    rgb[1] = x;
    rgb[2] = x;
}
```

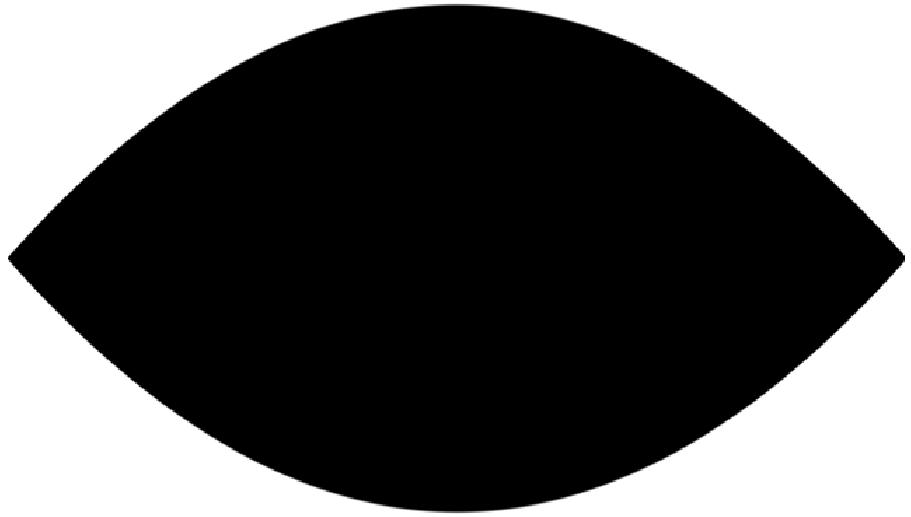
```
void draw( float *rgb, float x,
                       float y )
{
    rgb[0] = y;
    rgb[1] = y;
    rgb[2] = y;
}
```

```
void draw( float *rgb, float x,
                       float y )
{
    rgb[0] = (x+y)*0.7071f;
    rgb[1] = (x+y)*0.7071f;
    rgb[2] = (x+y)*0.7071f;
}
```

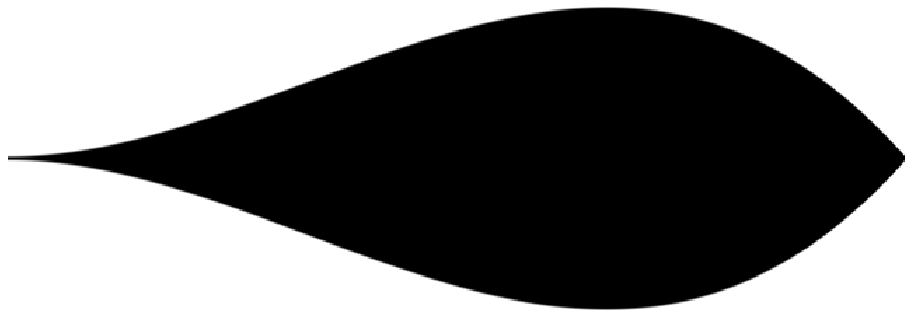
```
void draw( float *rgb, float x, float y )
{
    float rx = 2.0f*(x - 0.5f)*1.33f;
    float ry = 2.0f*(y - 0.5f);

    float h = x*(1.0f-x);

    float e = fabsf(ry) - h;

    float f = smoothstep( e, 0.0f, 0.01f );

    rgb[0] = f;
    rgb[1] = f;
    rgb[2] = f;
}
```



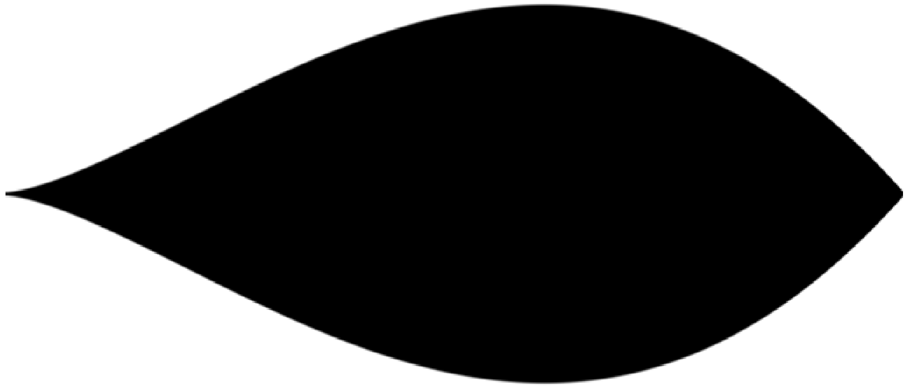
```
void draw( float *rgb, float x, float y )
{
    float rx = 2.0f*(x - 0.5f)*1.33f;
    float ry = 2.0f*(y - 0.5f);

    float h = 3.0f*x*(1.0f-x);

    float e = fabsf(ry) - h;

    float f = smoothstep( e, 0.0f, 0.01f );

    rgb[0] = f;
    rgb[1] = f;
    rgb[2] = f;
}
```



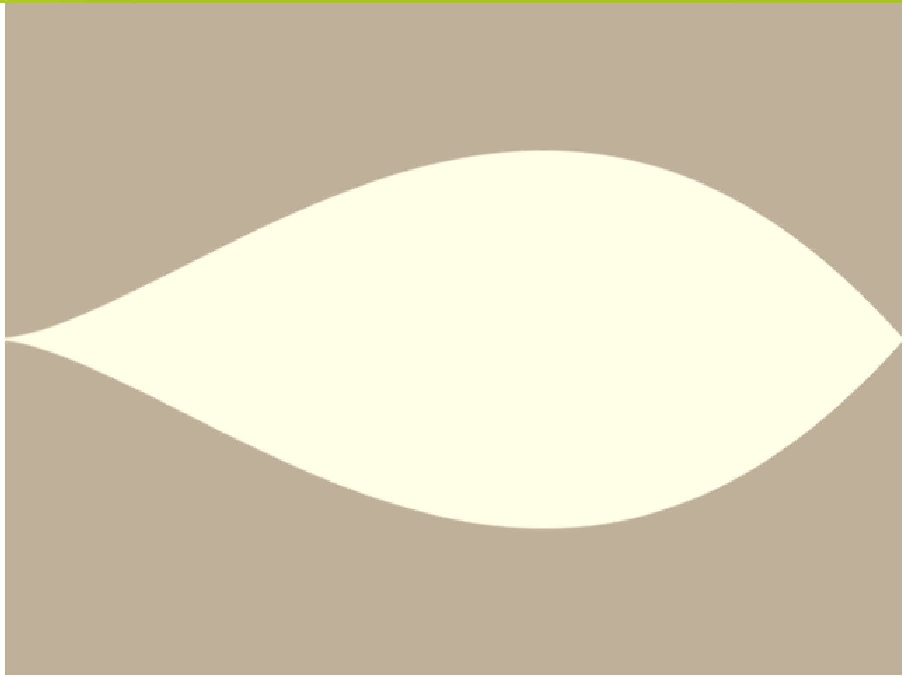
```
void draw( float *rgb, float x, float y )
{
    float rx = 2.0f*(x - 0.5f)*1.33f;
    float ry = 2.0f*(y - 0.5f);

    float h = 3.0f*x*x*(1.0f-x);

    float e = fabsf(ry) - h;

    float f = smoothstep( e, 0.0f, 0.01f );

    rgb[0] = f;
    rgb[1] = f;
    rgb[2] = f;
}
```



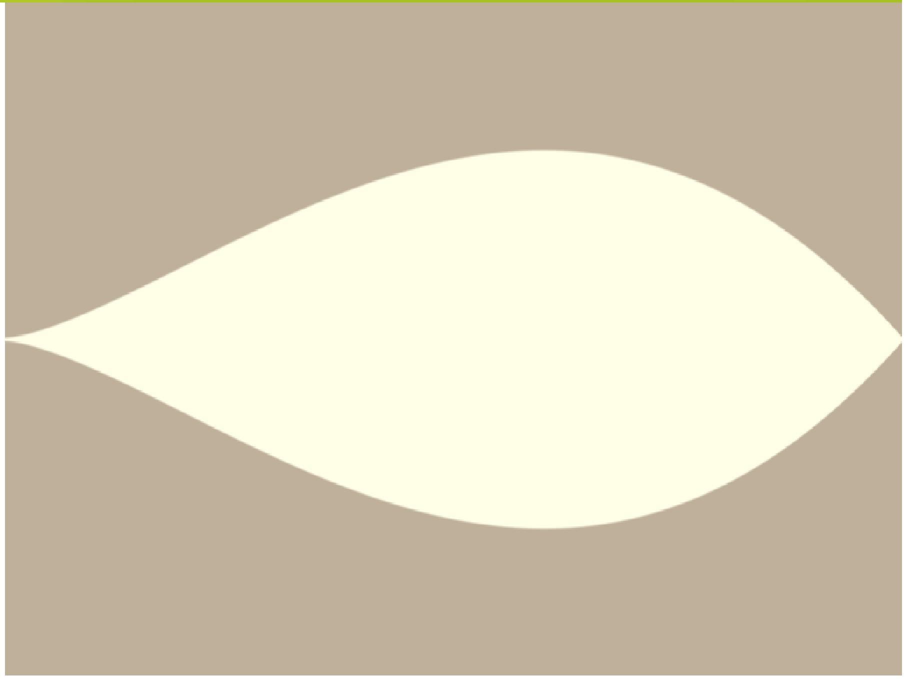
```
void draw( float *rgb, float x, float y )
{
    float rx = 2.0f*(x - 0.5f)*1.33f;
    float ry = 2.0f*(y - 0.5f);

    float h = 3.0f*sqrtf(x*x*x)*(1.0f-x);

    float e = fabsf(ry) - h;

    float f = smoothstep( e, 0.0f, 0.01f );

    rgb[0] = f;
    rgb[1] = f;
    rgb[2] = f;
}
```



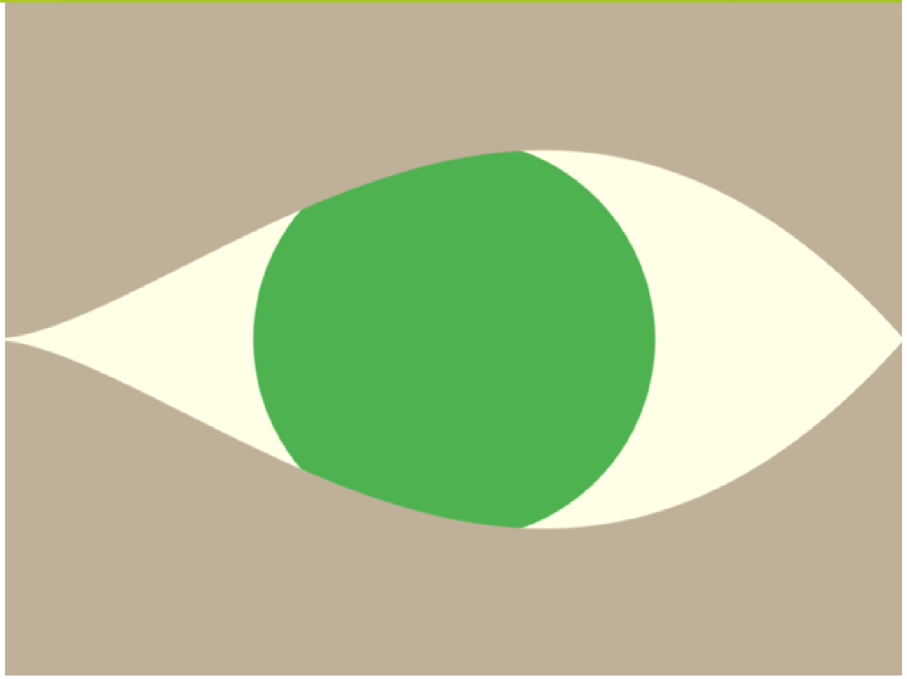
```
void draw( float *rgb, float x, float y )
{
    float rx = 2.0f*(x - 0.5f)*1.33f;
    float ry = 2.0f*(y - 0.5f);

    float h = 3.0f*sqrtf(x*x*x)*(1.0f-x);

    float e = fabsf(ry) - h;

    float f = smoothstep( e, 0.0f, 0.01f );

    float cPiel[3];
    float cOjo[3];

    eye(  cOjo,  x, y );
    skin( cPiel, x, y );

    collerp( rgb, f, cOjo, cPiel );
}
```



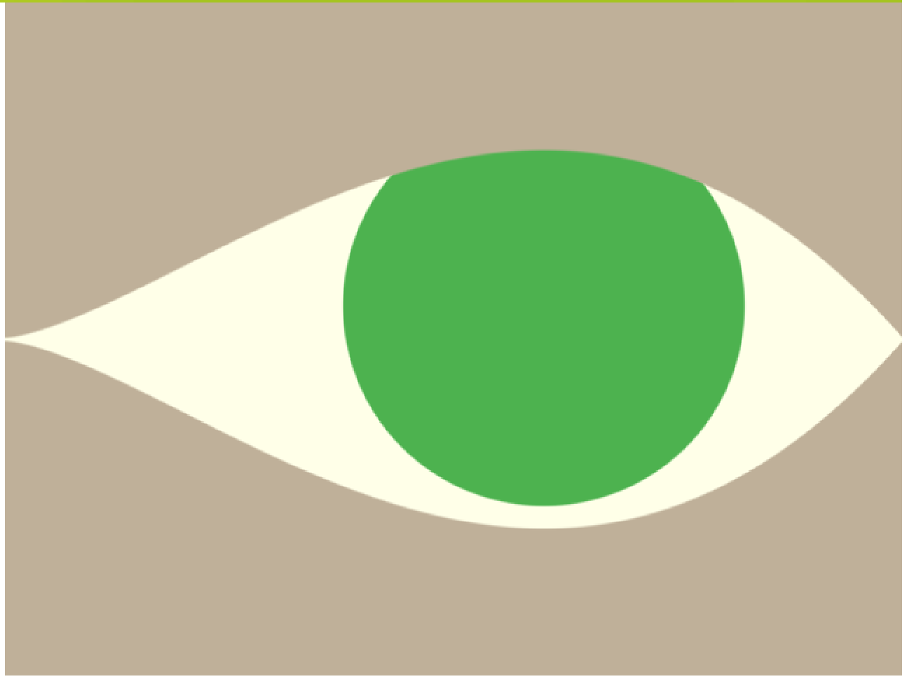
```
void eye( float *rgb, float x, float y, float b )
{
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};

    rgb[0] = fue[0];
    rgb[1] = fue[1];
    rgb[2] = fue[2];
}
```



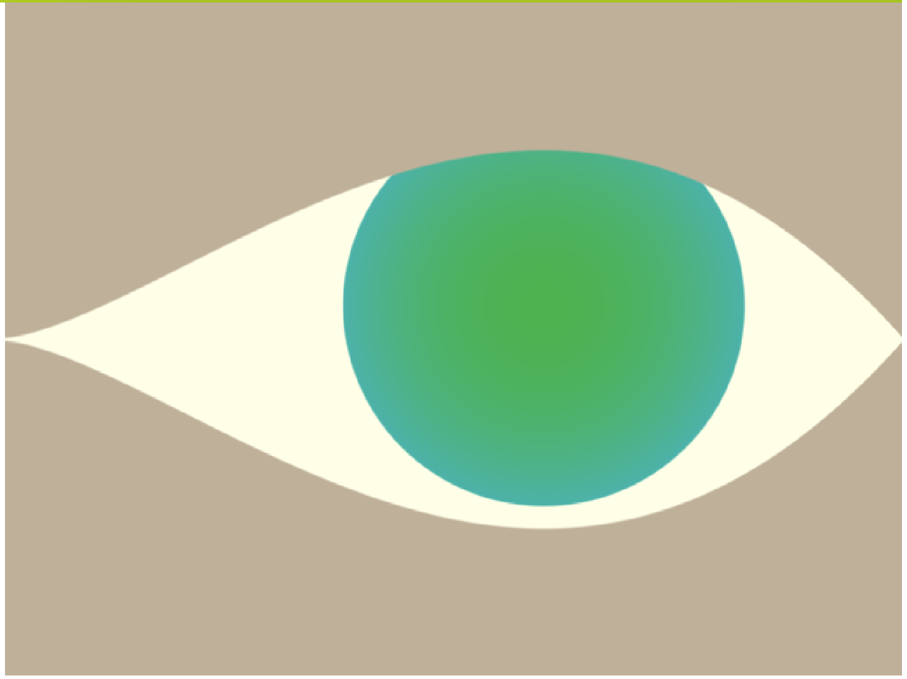
```
void eye( float *rgb, float x, float y, float b )
{
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f};

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);
}
```



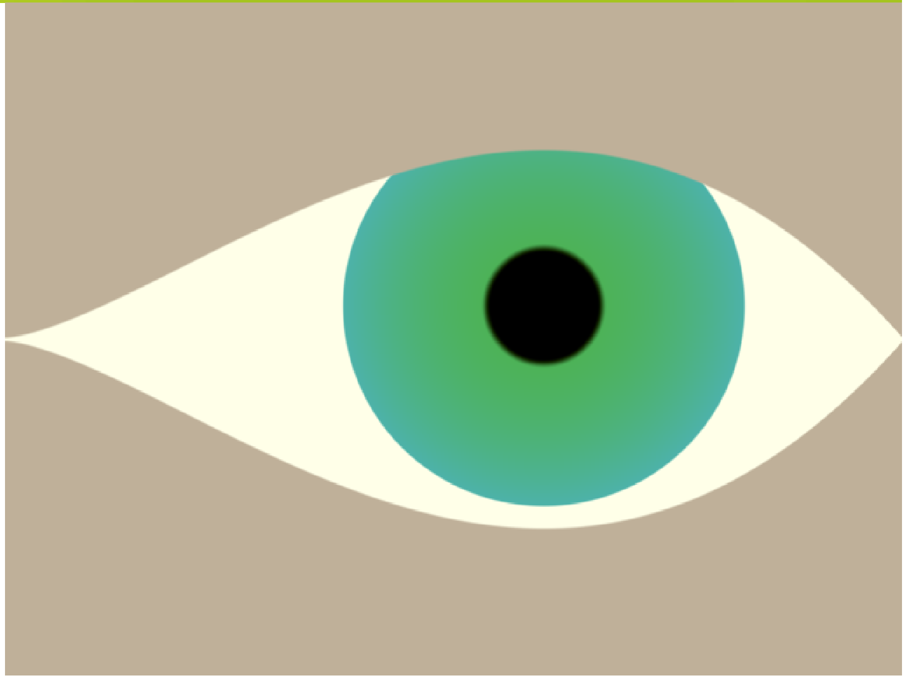
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f};

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);
}
```



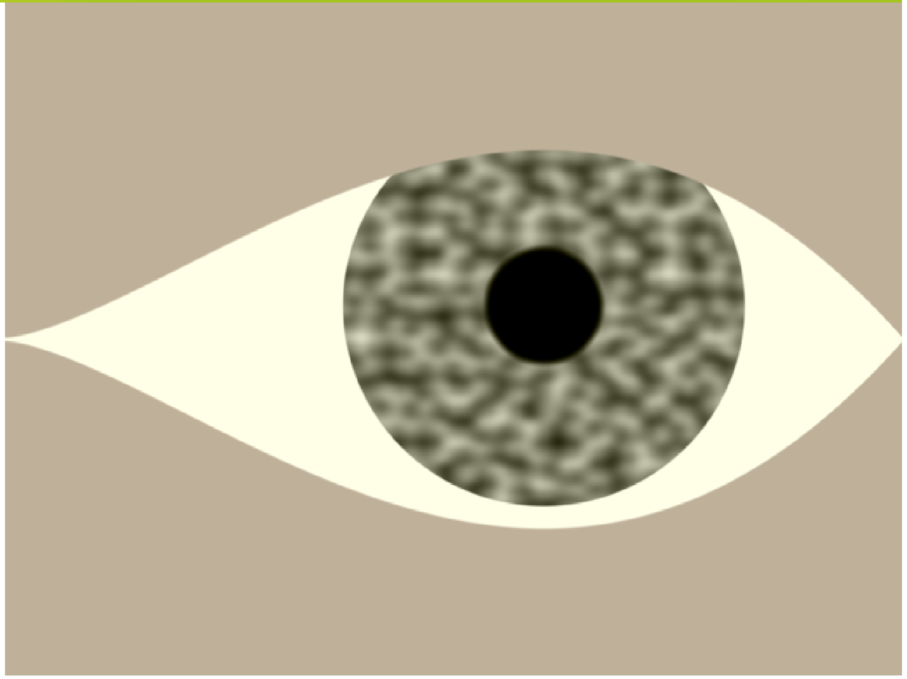
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);
}
```



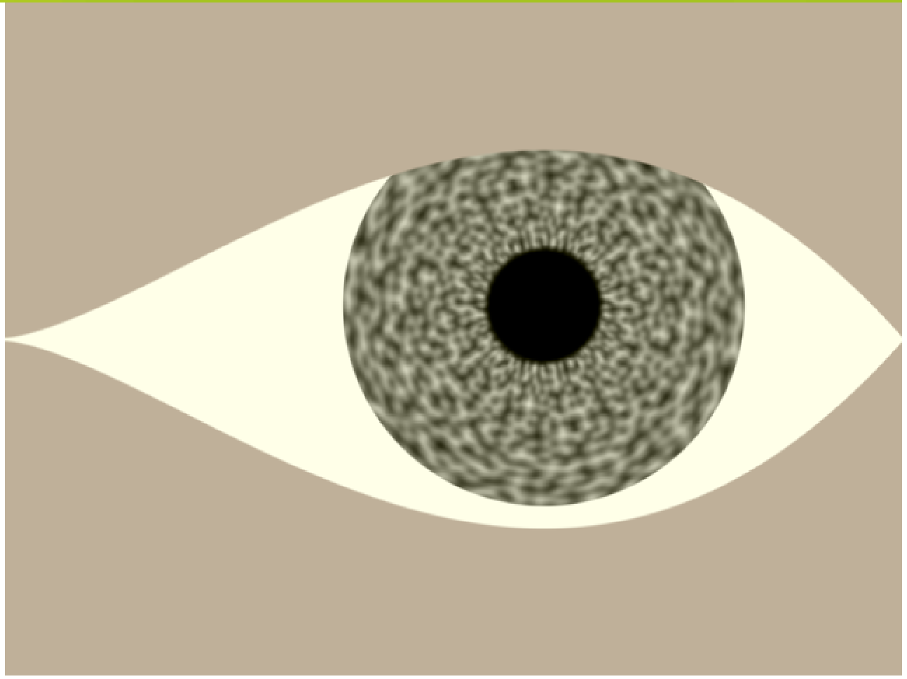
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);
}
```



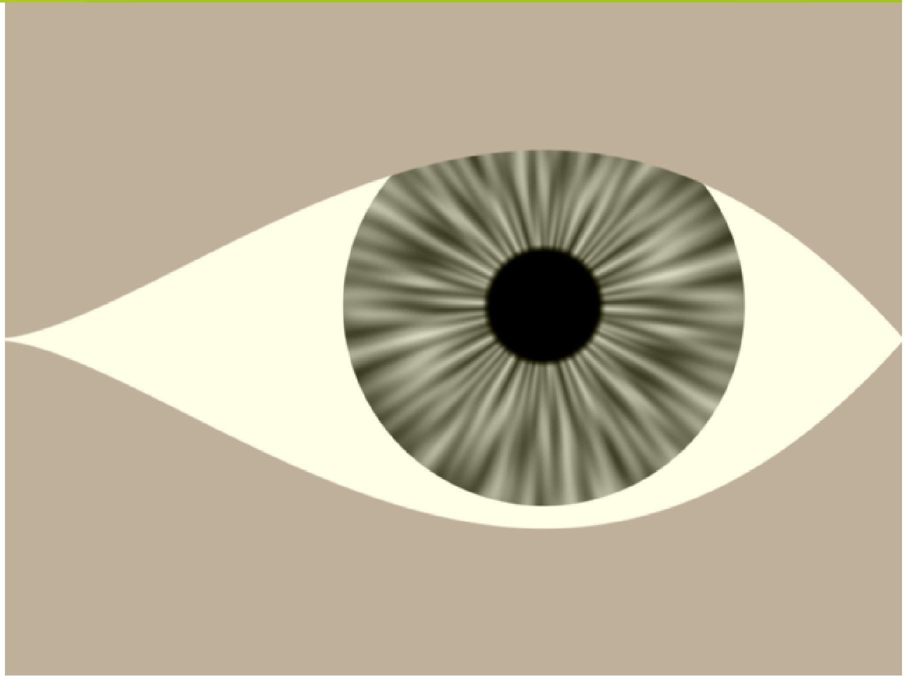
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.5f+0.5f*noise2f(32.0f*x, 32.0f*y,32,32);
    den[0] = no;
    den[1] = no;
    den[2] = no;

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);
}
```



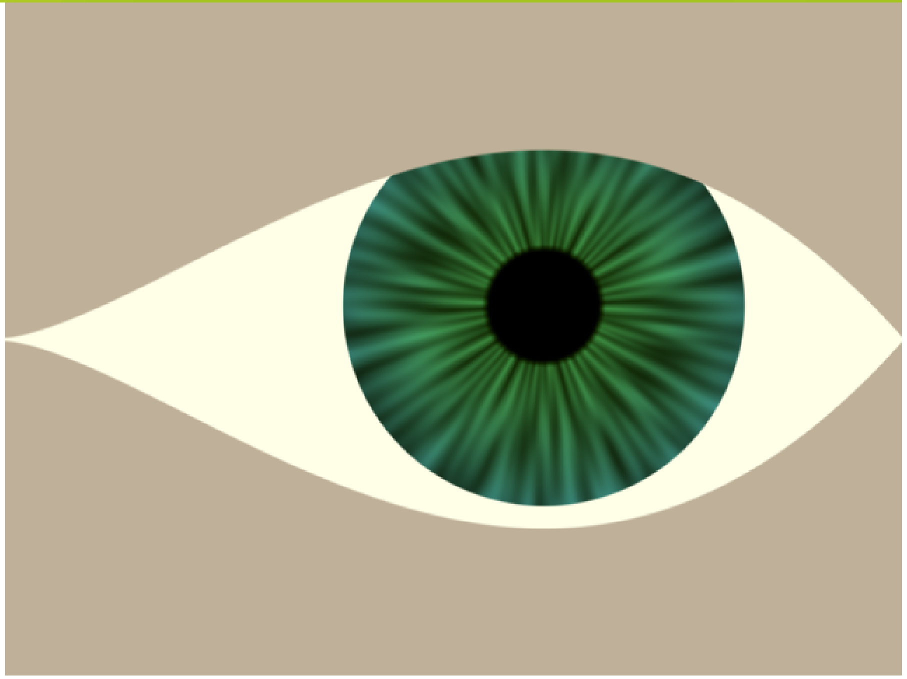
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.5f+0.5f*noise2f(32.0f*r, 32.0f*a/pi,32,32);
    den[0] = no;
    den[1] = no;
    den[2] = no;

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);
}
```



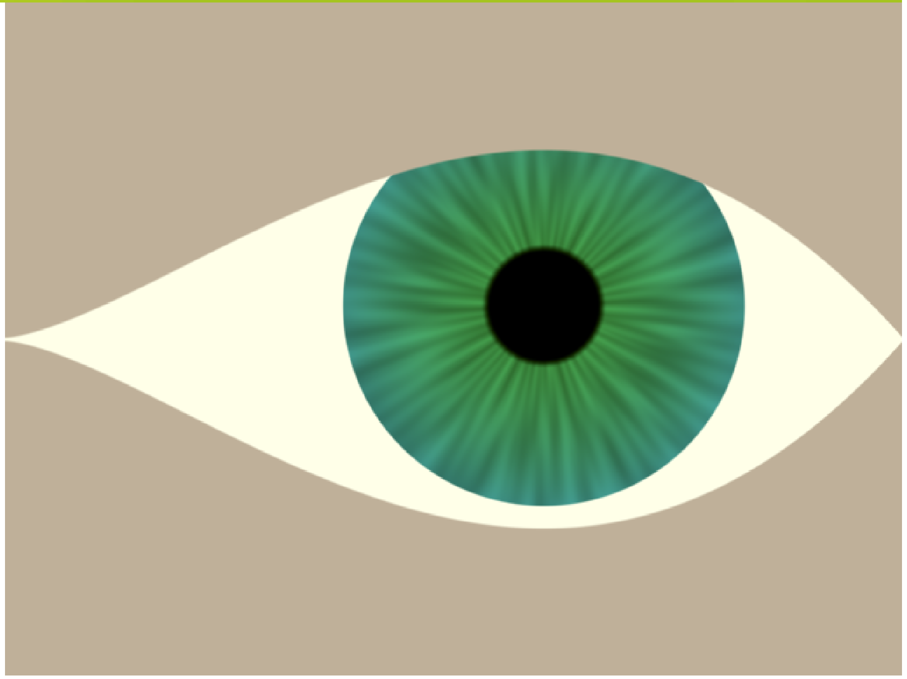
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.5f+0.5f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    den[0] = no;
    den[1] = no;
    den[2] = no;

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);
}
```



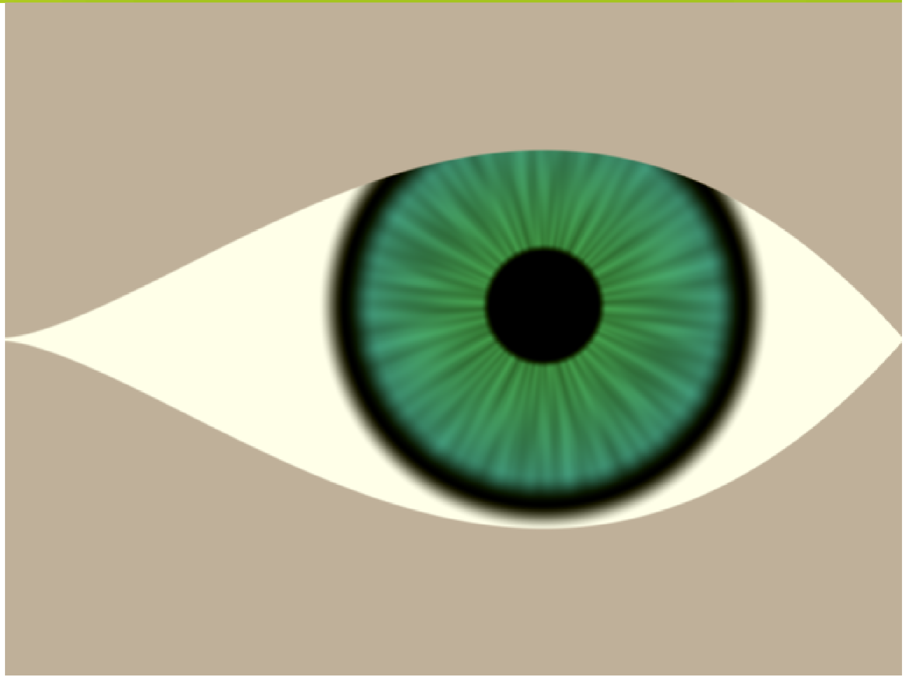
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.5f+0.5f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);
}
```



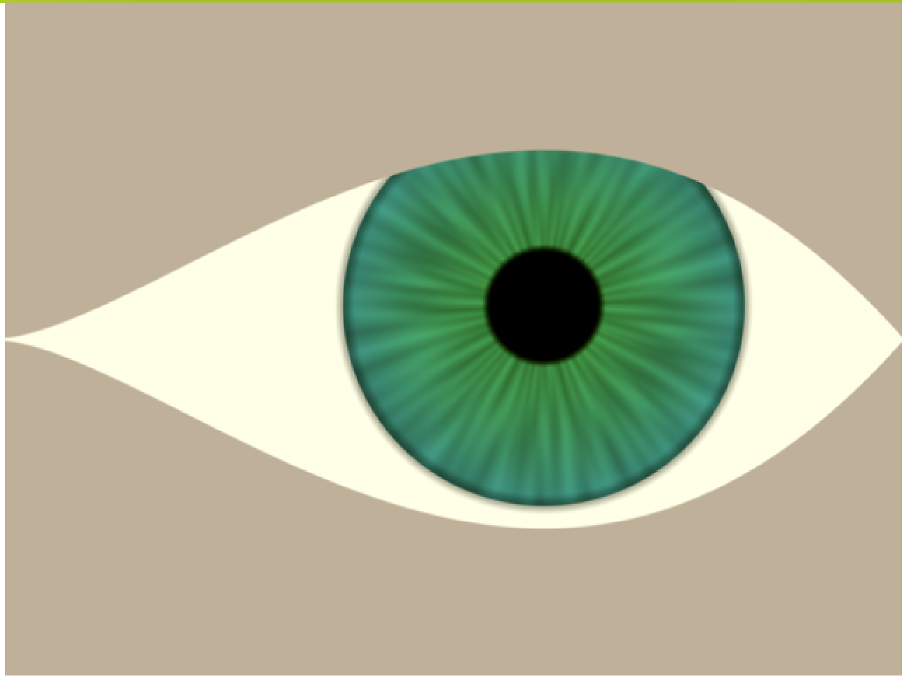
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);
}
```



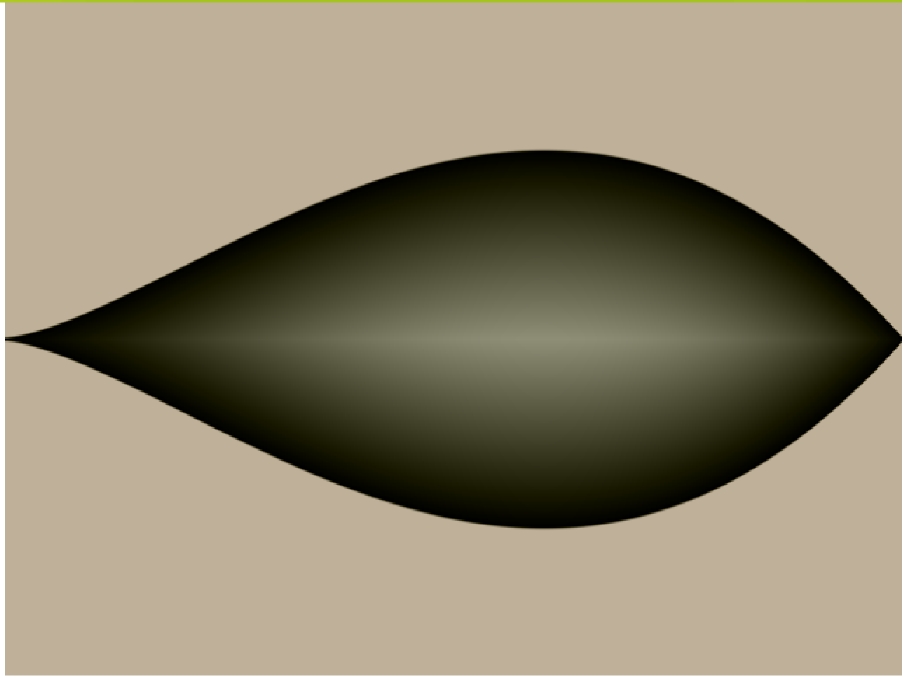
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.25f,.35f)-smoothstep(e,.35f,.45f);
    ri = 1.0f-ri;
    colsca(rgb, ri);
}
```



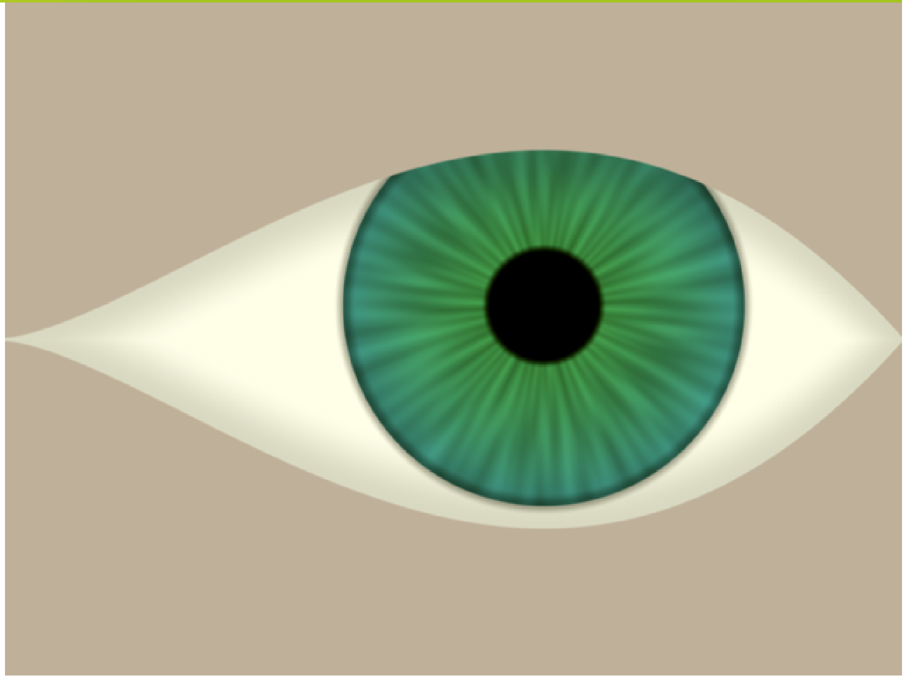
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);
}
```



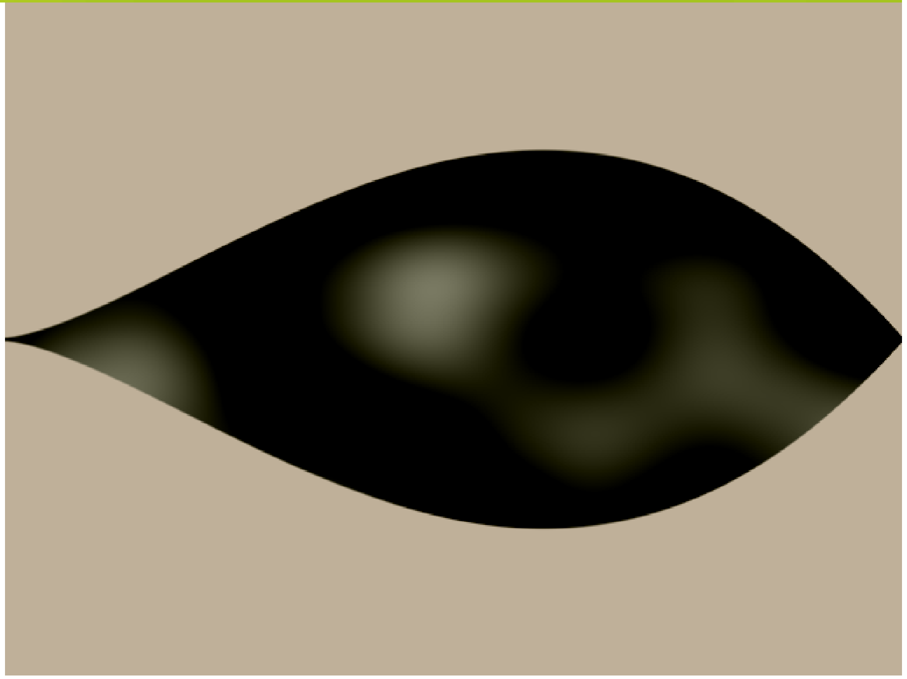
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // shadow
    rgb[0] = -b;
    rgb[1] = -b;
    rgb[2] = -b;
}
```



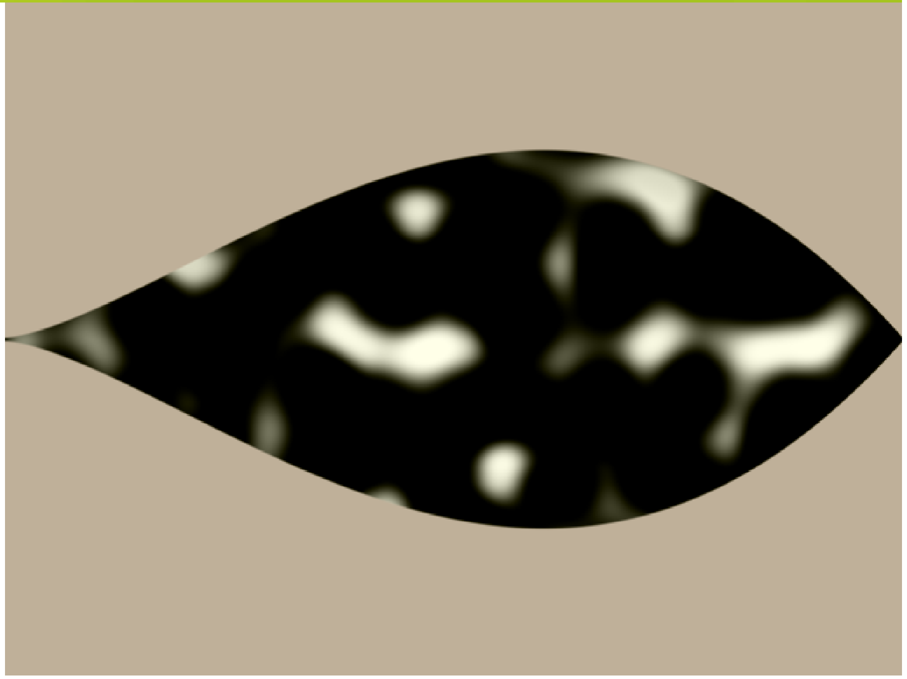
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



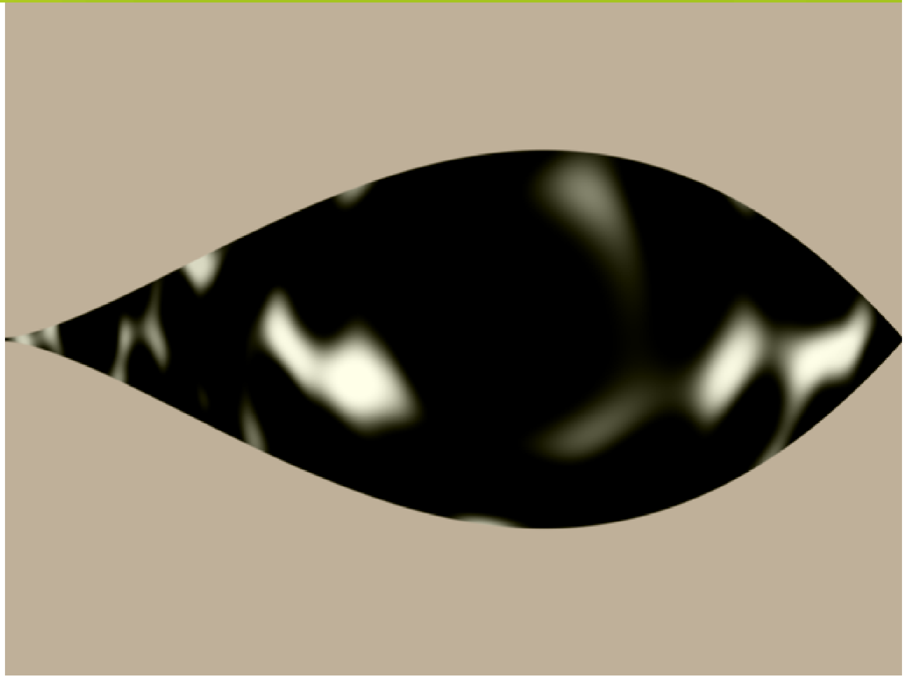
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r1 = r;
    float re = noise2f(2.0f+4.0f*r1*cosf(a),
                             4.0f*r1*sinf(a), 256, 256);
    rgb[0] = re;
    rgb[1] = re;
    rgb[2] = re;

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



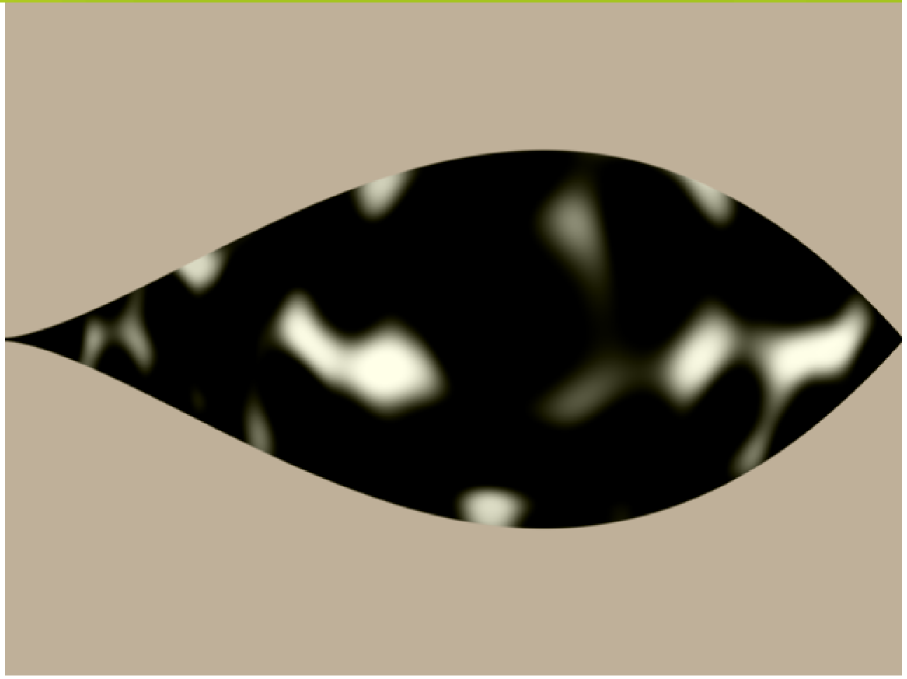
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r1 = r;
    float re = noise2f(2.0f+4.0f*r1*cosf(a),
                            4.0f*r1*sinf(a), 256, 256);
    re = smoothstep(re, 0.1f, 0.5f);
    rgb[0] = re;
    rgb[1] = re;
    rgb[2] = re;

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



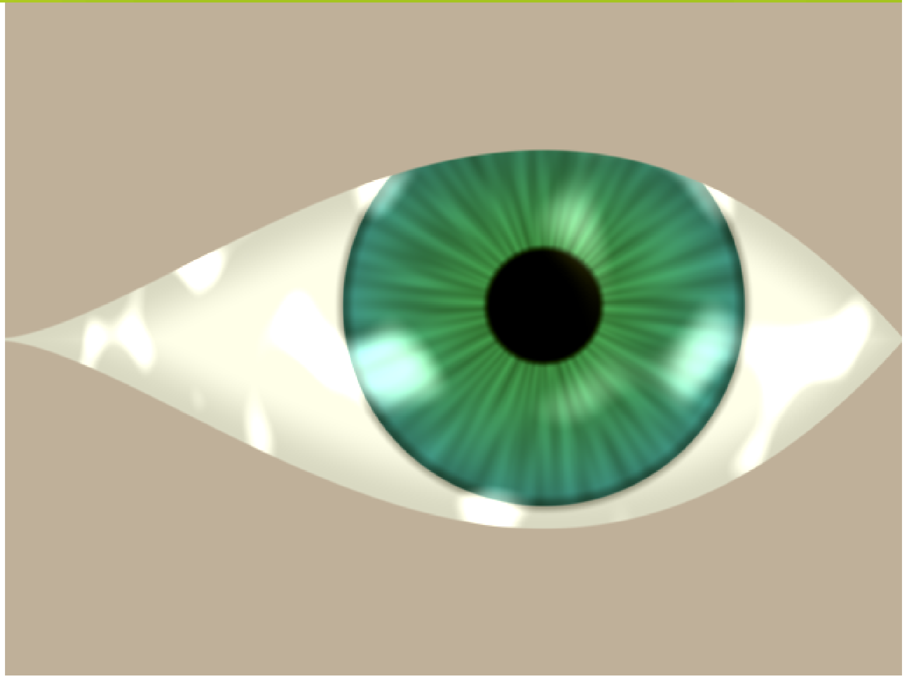
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r2 = r*r;
    float re = noise2f(2.0f+4.0f*r2*cosf(a),
                        4.0f*r2*sinf(a), 256, 256);
    re = smoothstep(re, 0.1f, 0.5f);
    rgb[0] = re;
    rgb[1] = re;
    rgb[2] = re;

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



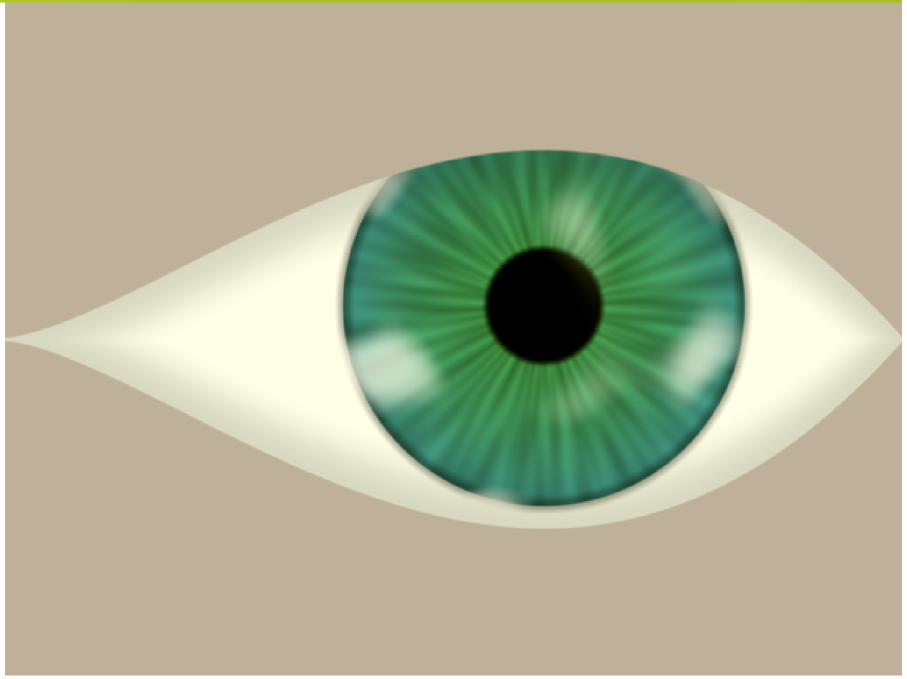
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r3 = sqrtf(r*r*r);
    float re = noise2f(2.0f+4.0f*r3*cosf(a),
                             4.0f*r3*sinf(a), 256, 256);
    re = smoothstep(re, 0.1f, 0.5f);
    rgb[0] = re;
    rgb[1] = re;
    rgb[2] = re;

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



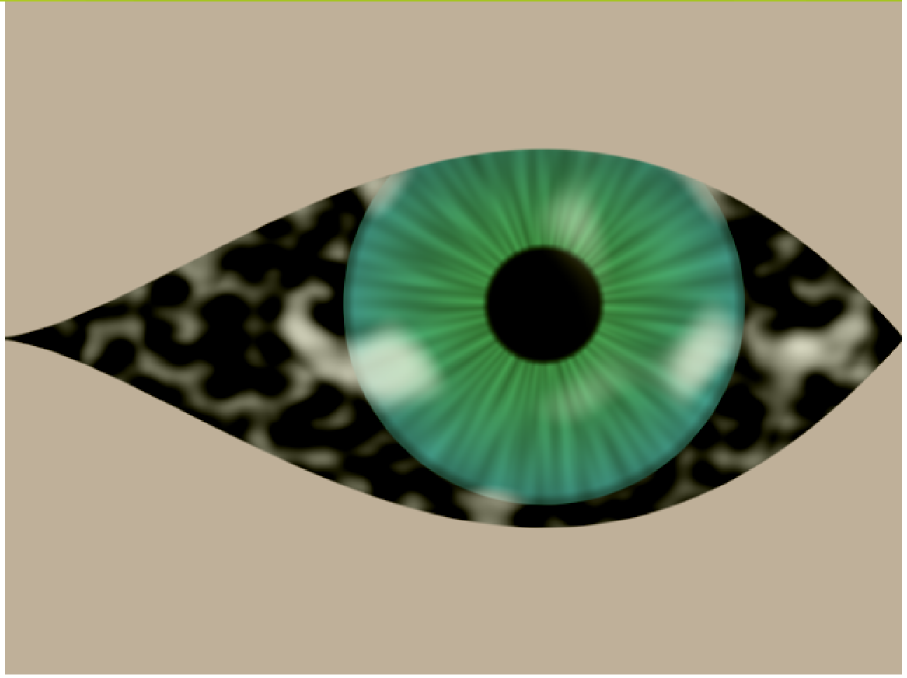
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e };

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r3 = sqrtf(r*r*r);
    float re = noise2f(2.0f+4.0f*r3*cosf(a),
                        4.0f*r3*sinf(a), 256, 256);
    re = 0.6f*smoothstep(re, 0.1f, 0.5f);
    rgb[0] += re;
    rgb[1] += re;
    rgb[2] += re;

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



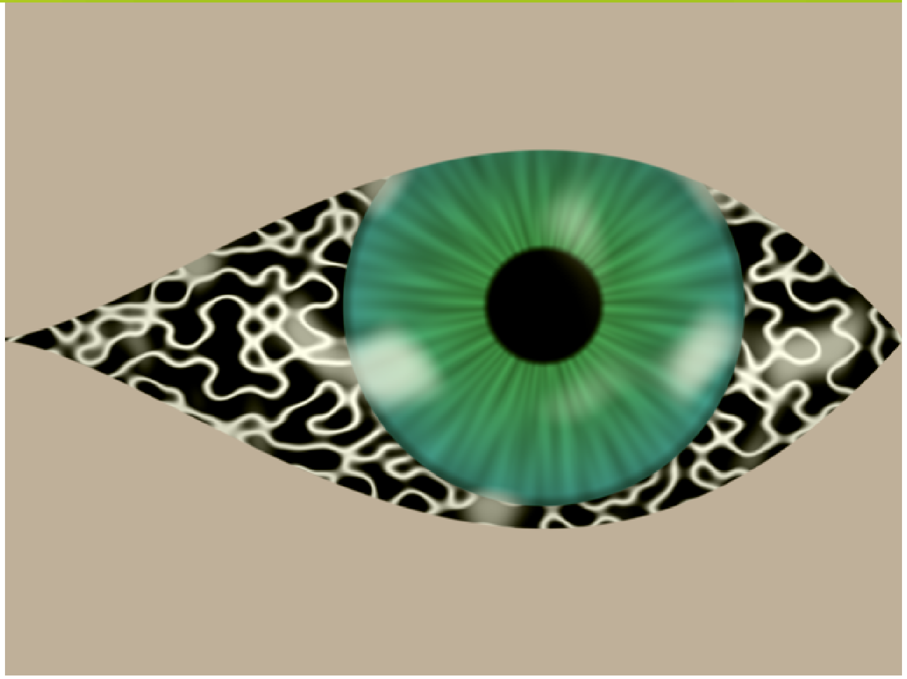
```
void eye( float *rgb, float x, float y, float b )
{
    y += 0.05f;
    x -= 0.10f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    float den[3] = {0.3f, 0.7f, 0.4f+e};

    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r3 = sqrtf(r*r*r);
    float re = noise2f(2.0f+4.0f*r3*cosf(a),
                        4.0f*r3*sinf(a), 256, 256);
    re = 0.8f*smoothstep(re, 0.1f, 0.5f);
    rgb[0] += re*(1.0f-rgb[0]);
    rgb[1] += re*(1.0f-rgb[1]);
    rgb[2] += re*(1.0f-rgb[2]);

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



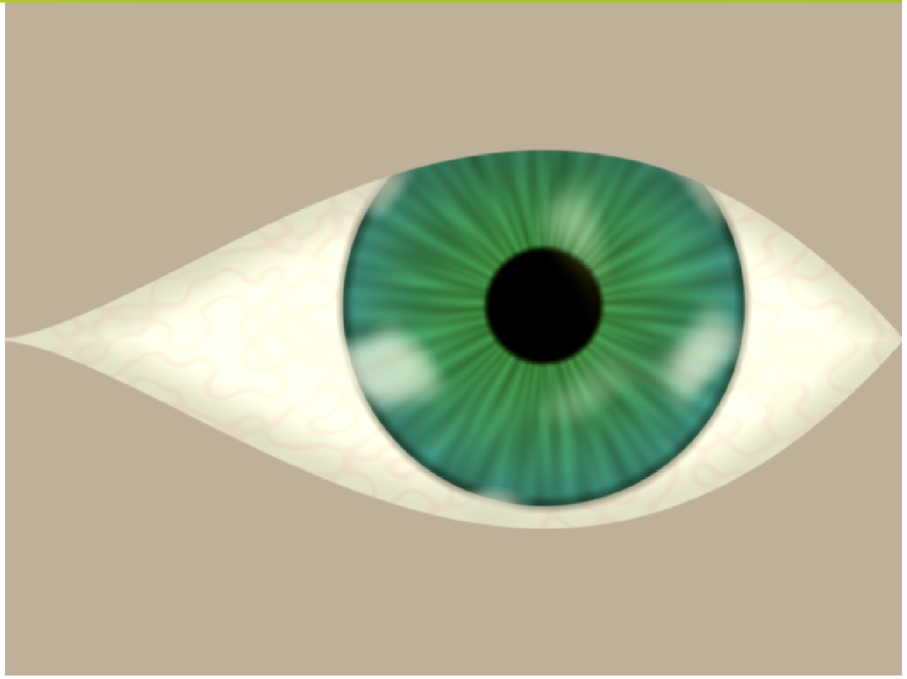
```
void eye( float *rgb, float x, float y, float b )
{
    x -= 0.10f; y += 0.05f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    // blood
    float ven = noise2f(24.0f*x, 24.0f*y, 256, 256);
    fue[0] = ven;
    fue[1] = ven;
    fue[2] = ven;

    float den[3] = {0.3f, 0.7f, 0.4f+e};
    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r3 = sqrtf(r*r*r);
    float re = noise2f(2.0f+4.0f*r3*cosf(a),
                       4.0f*r3*sinf(a), 256, 256);
    re = 0.8f*smoothstep(re, 0.1f, 0.5f);
    rgb[0] += re*(1.0f-rgb[0]);
    rgb[1] += re*(1.0f-rgb[1]);
    rgb[2] += re*(1.0f-rgb[2]);

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



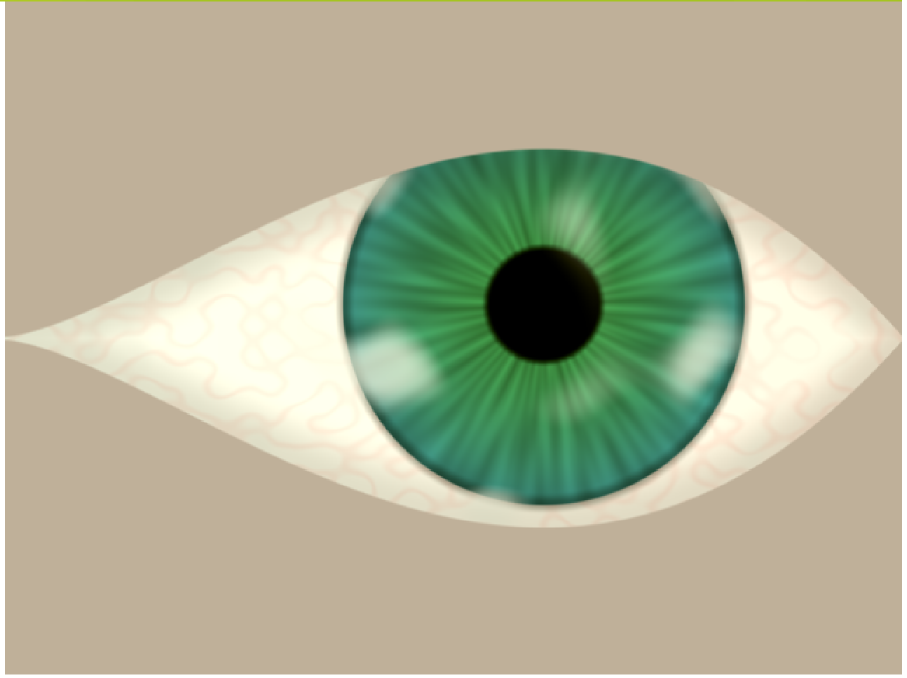
```
void eye( float *rgb, float x, float y, float b )
{
    x -= 0.10f; y += 0.05f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    // blood
    float ven = noise2f(24.0f*x, 24.0f*y, 256, 256);
    ven = smoothstep(ven,-.2f,.0f)-smoothstep(ven,.0f,.2f);
    fue[0] = ven;
    fue[1] = ven;
    fue[2] = ven;

    float den[3] = {0.3f, 0.7f, 0.4f+e};
    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r3 = sqrtf(r*r*r);
    float re = noise2f(2.0f+4.0f*r3*cosf(a),
                       4.0f*r3*sinf(a), 256, 256);
    re = 0.8f*smoothstep(re, 0.1f, 0.5f);
    rgb[0] += re*(1.0f-rgb[0]);
    rgb[1] += re*(1.0f-rgb[1]);
    rgb[2] += re*(1.0f-rgb[2]);

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



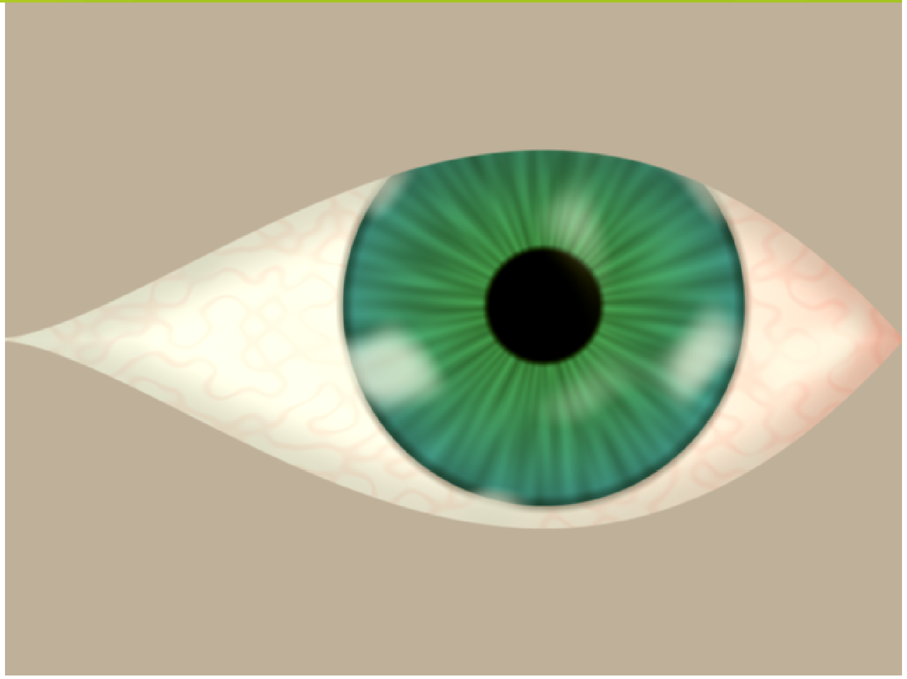
```
void eye( float *rgb, float x, float y, float b )
{
    x -= 0.10f; y += 0.05f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    // blood
    float ven = noise2f(24.0f*x, 24.0f*y, 256, 256);
    ven = smoothstep(ven,-.2f,.0f)-smoothstep(ven,.0f,.2f);
    fue[0] = fue[0] + 0.04f - 0.00f*ven;
    fue[1] = fue[1] + 0.04f - 0.05f*ven;
    fue[2] = fue[2] + 0.04f - 0.05f*ven;

    float den[3] = {0.3f, 0.7f, 0.4f+e};
    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r3 = sqrtf(r*r*r);
    float re = noise2f(2.0f+4.0f*r3*cosf(a),
                        4.0f*r3*sinf(a), 256, 256);
    re = 0.8f*smoothstep(re, 0.1f, 0.5f);
    rgb[0] += re*(1.0f-rgb[0]);
    rgb[1] += re*(1.0f-rgb[1]);
    rgb[2] += re*(1.0f-rgb[2]);

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



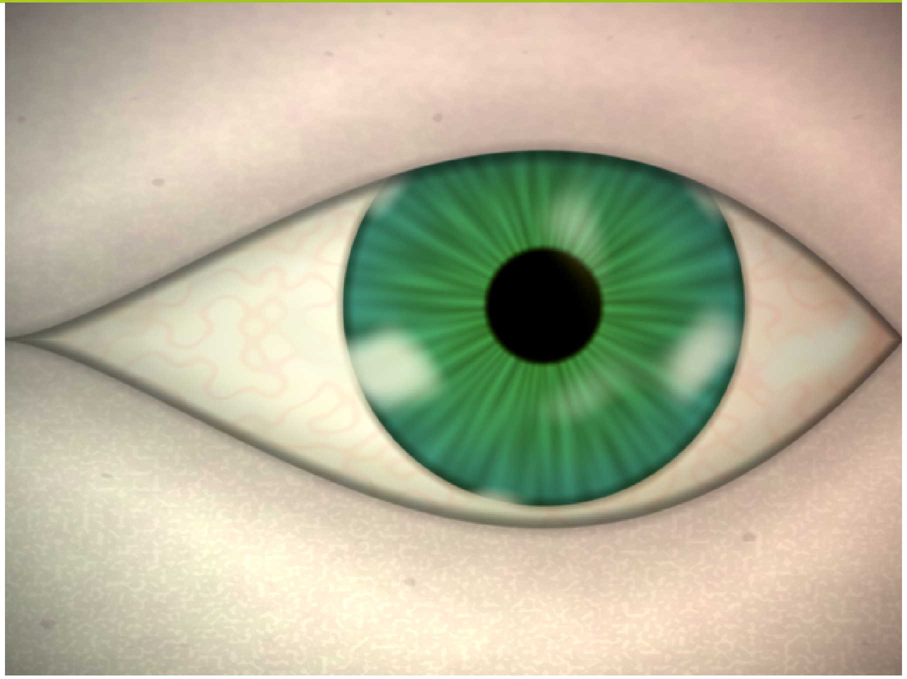
```
void eye( float *rgb, float x, float y, float b )
{
    x -= 0.10f; y += 0.05f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    // blood
    float ven = noise2f(24.0f*x, 24.0f*y, 256, 256);
    ven = smoothstep(ven,-.2f,.0f)-smoothstep(ven,.0f,.2f);
    ven += x;
    fue[0] = fue[0] + 0.04f - 0.00f*ven;
    fue[1] = fue[1] + 0.04f - 0.05f*ven;
    fue[2] = fue[2] + 0.04f - 0.05f*ven;

    float den[3] = {0.3f, 0.7f, 0.4f+e};
    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r3 = sqrtf(r*r*r);
    float re = noise2f(2.0f+4.0f*r3*cosf(a),
                       4.0f*r3*sinf(a), 256, 256);
    re = 0.8f*smoothstep(re, 0.1f, 0.5f);
    rgb[0] += re*(1.0f-rgb[0]);
    rgb[1] += re*(1.0f-rgb[1]);
    rgb[2] += re*(1.0f-rgb[2]);

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



```
void eye( float *rgb, float x, float y, float b )
{
    x -= 0.10f; y += 0.05f;
    float rx = 2.0f*(x-0.5f)*4.0f/3.0f;
    float ry = 2.0f*(y-0.5f);
    float a  = atan2f(ry, rx);
    float e  = rx*rx + ry*ry;
    float r  = sqrtf(e);

    float fue[3] = {1.0f, 1.0f, 1.0f};
    // blood
    float ven = noise2f(24.0f*x, 24.0f*y, 256, 256);
    ven = smoothstep(ven,-.2f,.0f)-smoothstep(ven,.0f,.2f);
    ven += x + x*x*x*x*x*x*7.0f;
    fue[0] = fue[0] + 0.04f - 0.00f*ven;
    fue[1] = fue[1] + 0.04f - 0.05f*ven;
    fue[2] = fue[2] + 0.04f - 0.05f*ven;

    float den[3] = {0.3f, 0.7f, 0.4f+e};
    float no = 0.8f+0.2f*noise2f(4.0f*r, 32.0f*a/pi,32,32);
    colsca(den, no);

    float f2 = smoothstep(e, 0.025f, 0.035f);
    colsca(den, f2);

    // blend in/out
    collerp(rgb, smoothstep(e, 0.35f, 0.36f), den, fue);

    // ring
    float ri=smoothstep(e,.31f,.35f)-smoothstep(e,.35f,.39f);
    ri = 1.0f-0.35f*ri;
    colsca(rgb, ri);

    // reflecion
    float r3 = sqrtf(r*r*r);
    float re = noise2f(2.0f+4.0f*r3*cosf(a),
                       4.0f*r3*sinf(a), 256, 256);
    re = 0.8f*smoothstep(re, 0.1f, 0.5f);
    rgb[0] += re*(1.0f-rgb[0]);
    rgb[1] += re*(1.0f-rgb[1]);
    rgb[2] += re*(1.0f-rgb[2]);

    // shadow
    colsca(rgb, 0.85f+0.15f*smoothstep(-b, 0.0f, 0.2f) );
}
```



```
void skin( float *rgb, float x, float y )
{
    float rx = 2.0f*(x - 0.5f)*1.33f;
    float ry = 2.0f*(y - 0.5f);

    float carne[3] = { 0.75f, 0.69f, 0.6f };

    float cel =0.95f+0.05f*noise2f(64.0f*x, 64.0f*y,256,256);
    col_sca( carne, cel );
    carne[0] += 0.03f*rx;
    carne[1] += 0.03f*ry;
    col_sca( carne, y*0.1f+0.9f );

    float bri = noise2f( 128.0f*x, 128.0f*y, 256, 256 );
    bri = 0.2f+0.8f*smoothstep( bri, 0.0f, 0.3f );
    col_finc( carne, bri*0.08f*y );

    float san  = 0.50f*noise2f(16.0f*x, 16.0f*y, 256, 256);
          san += 0.25f*noise2f(32.0f*x, 32.0f*y, 256, 256);
    carne[1] *= 1.0f-0.1f*san;

    float osc =  0.500f*noise2f(12.0f*x, 12.0f*y, 256, 256);
          osc += 0.250f*noise2f(24.0f*x, 24.0f*y, 256, 256);
          osc += 0.125f*noise2f(48.0f*x, 48.0f*y, 256, 256);
    col_sca( carne, 0.9f+0.1f*osc );

    carne[0] += 0.08f*x;
    carne[1] += 0.01f;

    float pecas = noise2f(32.0f*x, 32.0f*y, 256, 256);
    pecas = smoothstep( pecas, 0.48f, 0.6f );
    carne[0] *= 1.0f-0.15f*pecas;
    carne[1] *= 1.0f-0.17f*pecas;
    carne[2] *= 1.0f-0.17f*pecas;

    col_finc( carne, 0.14f );

    rgb[0] = carne[0]*1.25f+0.3f;
    rgb[1] = carne[1]*1.22f+0.3f;
    rgb[2] = carne[2]*1.20f+0.3f;
}
```



Index

2.5D procedural drawing

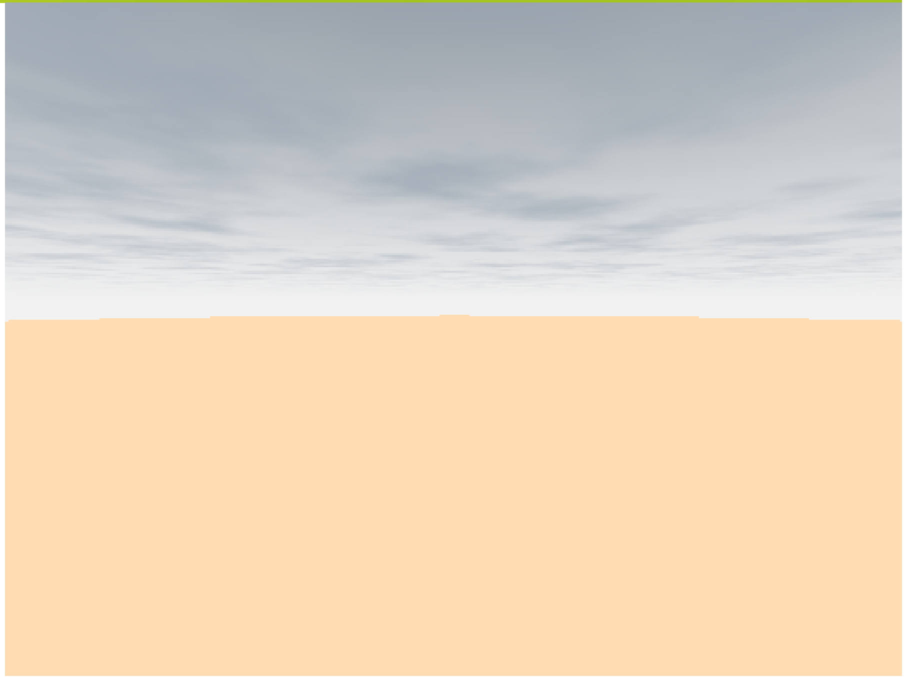
• Drawing in 2.5D (a heighmap) is basically the same as drawing in 2D, just easier.

• You can implement a raymarcher (like in the old voxel engines) to transform the heightmap into an image.

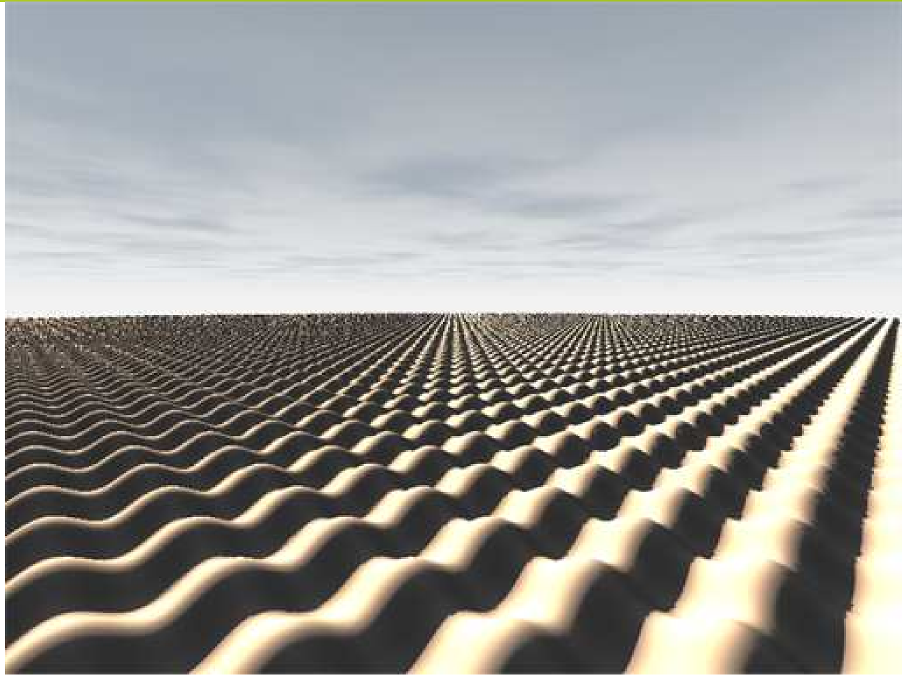
• In 2.5D it's easier to get closer to photorealism (it you are interested in it), cause

- You don't have to draw the shading, it is computed automatically by the usual ilumination models (lambert, shadows, scattering, etc).
- You don't have to fake perspective.

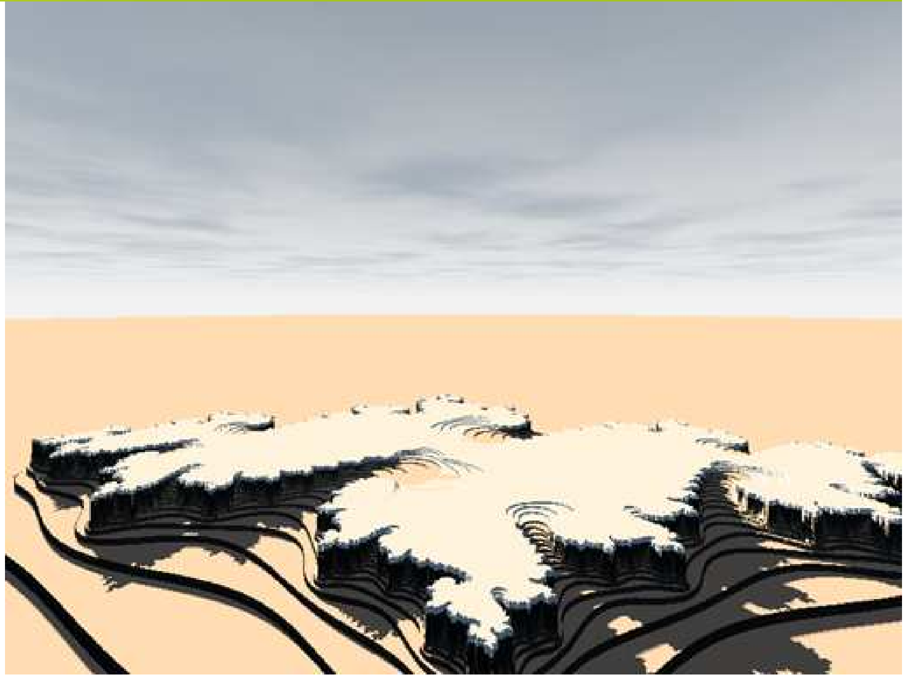
```
float map(float x, float z )
{
    return -1.2f;
}
```



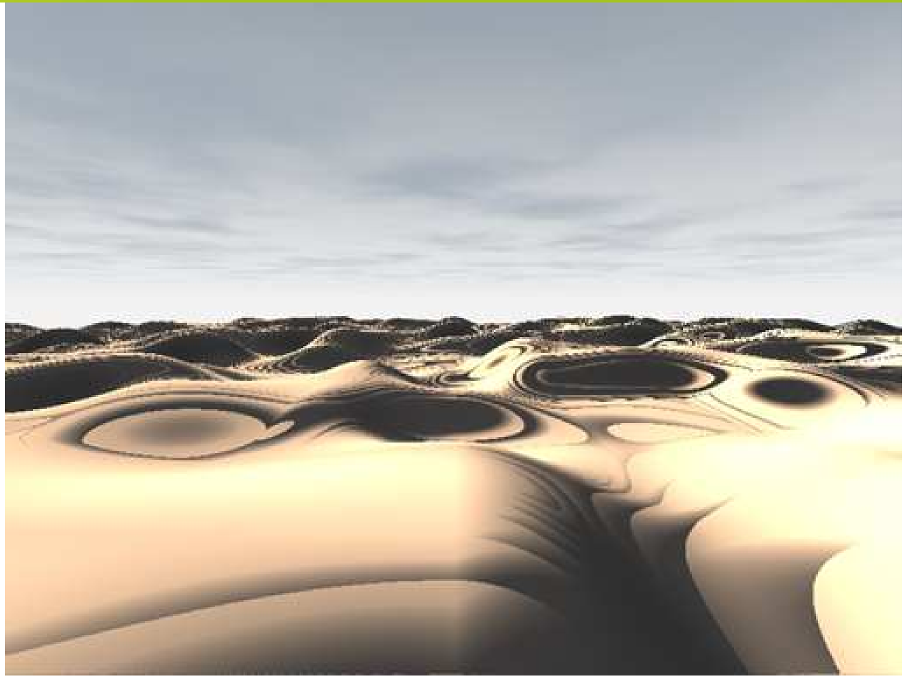
```
float map(float x, float z )
{
    return -1.2f + 0.1f*sinf(10.0f*x)*cosf(10.0f*z);
}
```



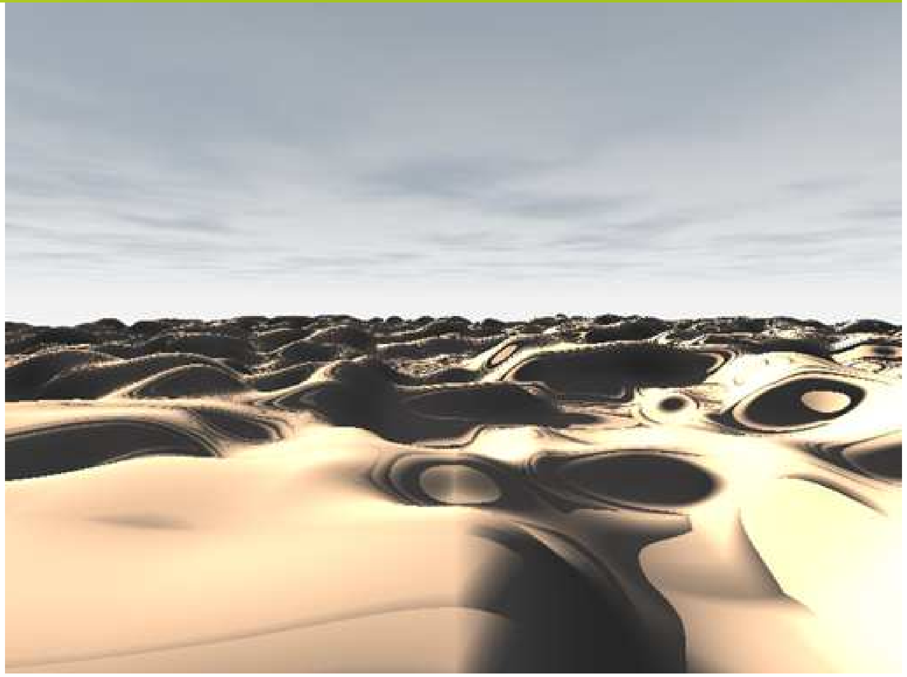
```
float map(float x, float z )
{
    int i;
    z -= 1.5f;
    for( i=0; i<20; i++ )
    {
        float nx = x*x-z*z  - 0.745f;
        float nz = 2.0f*x*z + 0.186f;
        if( nx*nx+nz*nz>4.0f ) break;
        x = nx;
        z = nz;
    }
    return -1.2f + sqrtf(sqrtf(sqrtf(i*0.001f)));
}
```



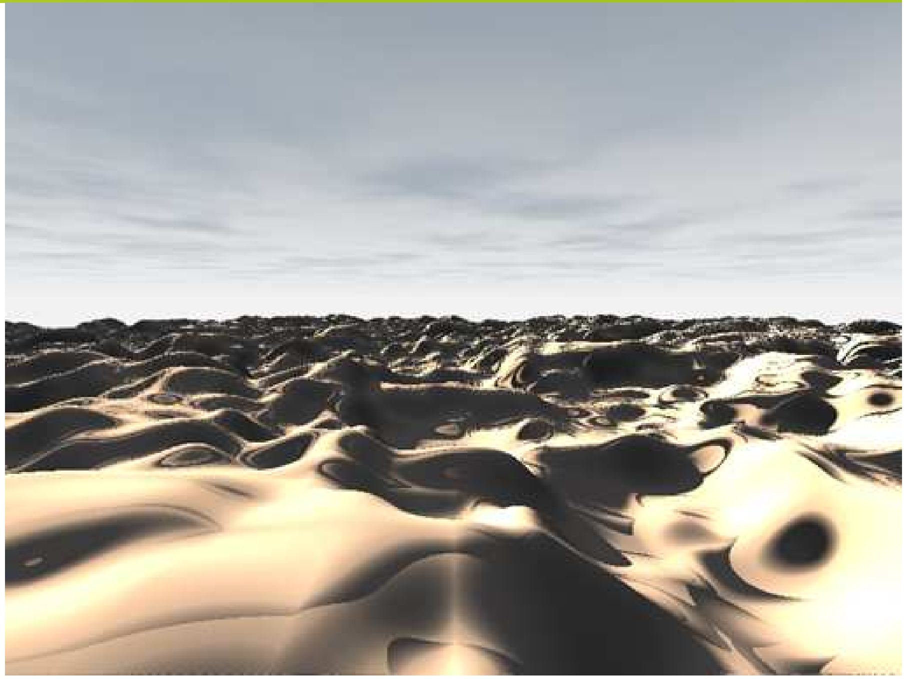
```
float map(float x, float z )
{
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;
    return f;
}
```



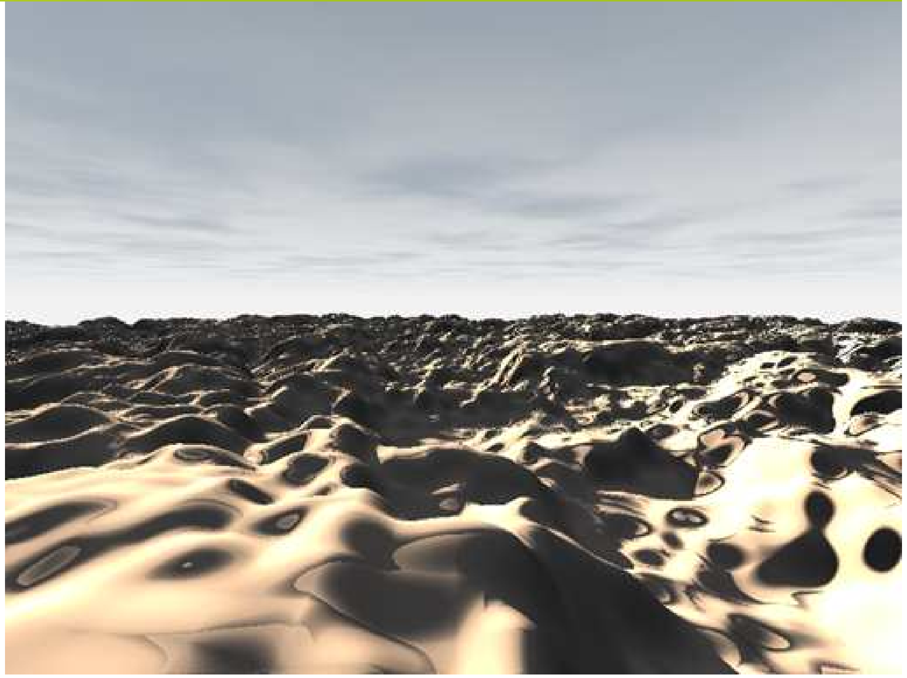
```
float map(float x, float z )
{
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;
    return f;
}
```



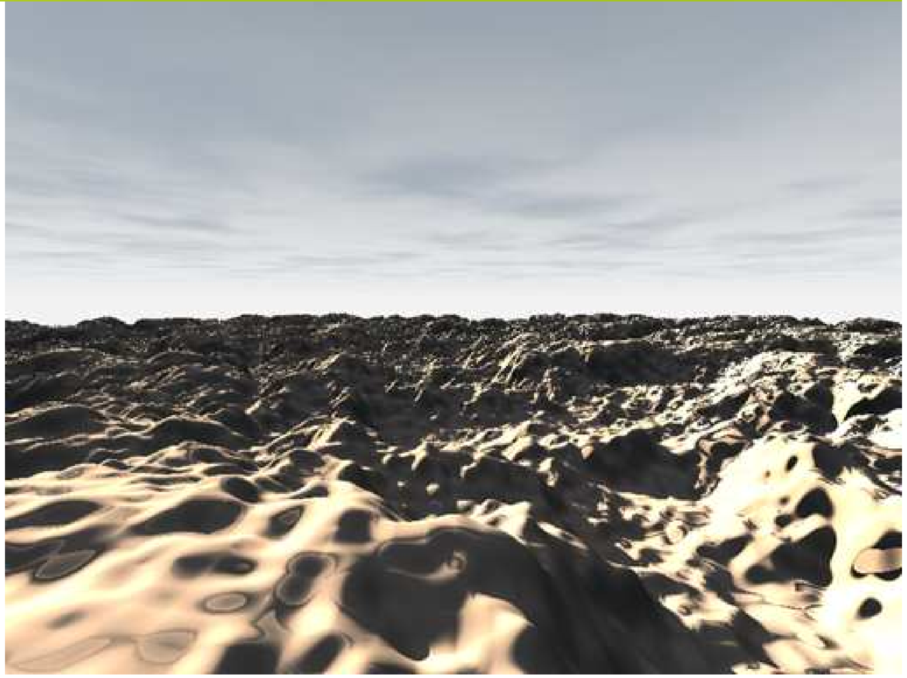
```
float map(float x, float z )
{
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f += 0.1250000f*noise2f( 4.0f*x,  4.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;
    return f;
}
```



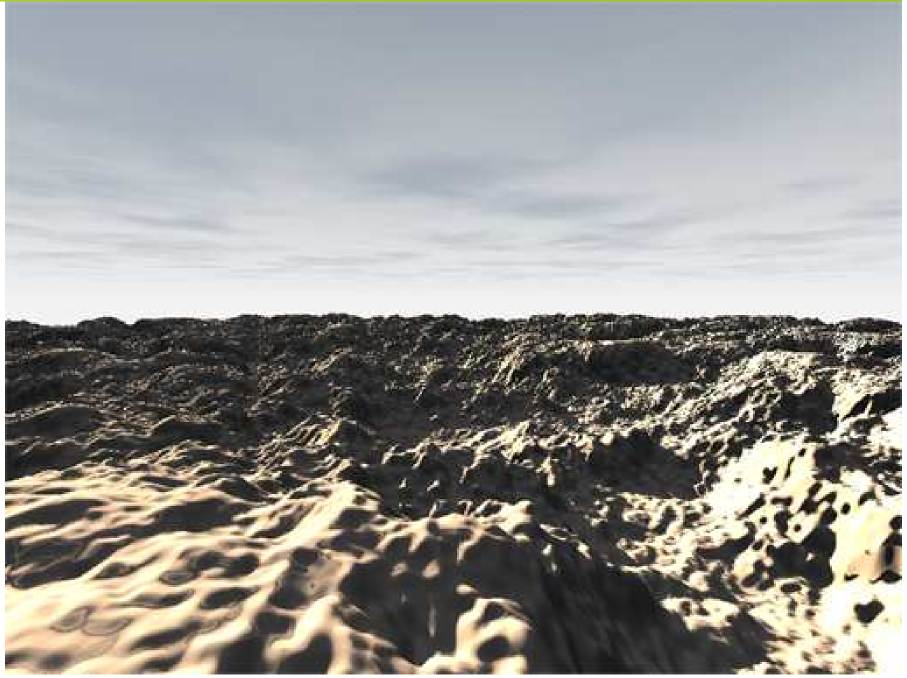
```
float map(float x, float z )
{
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f += 0.1250000f*noise2f( 4.0f*x,  4.0f*z);
    f += 0.0625000f*noise2f( 8.0f*x,  8.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;
    return f;
}
```



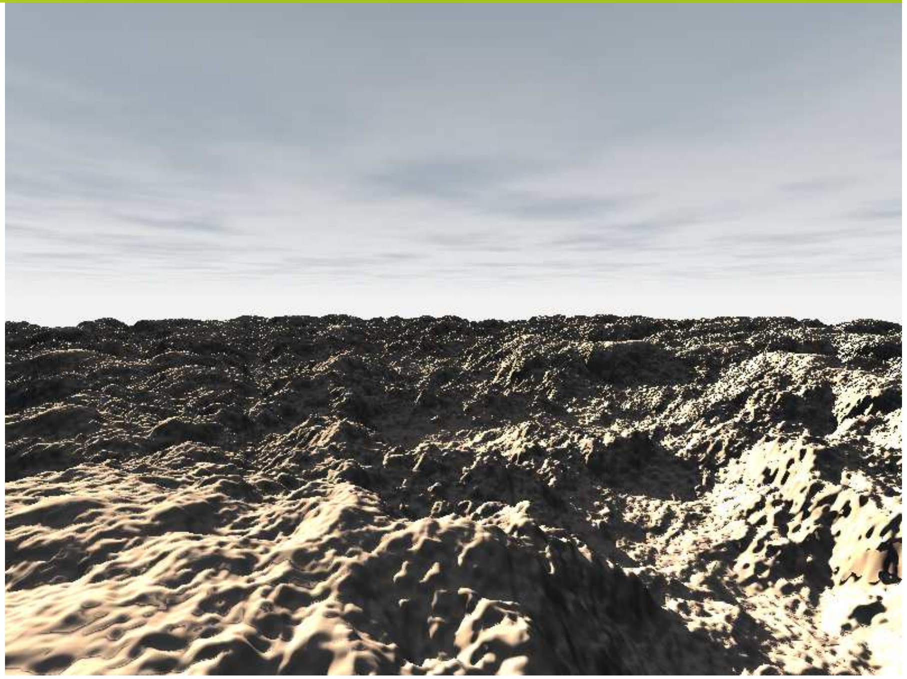
```
float map(float x, float z )
{
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f += 0.1250000f*noise2f( 4.0f*x,  4.0f*z);
    f += 0.0625000f*noise2f( 8.0f*x,  8.0f*z);
    f += 0.0312500f*noise2f(16.0f*x, 16.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;
    return f;
}
```



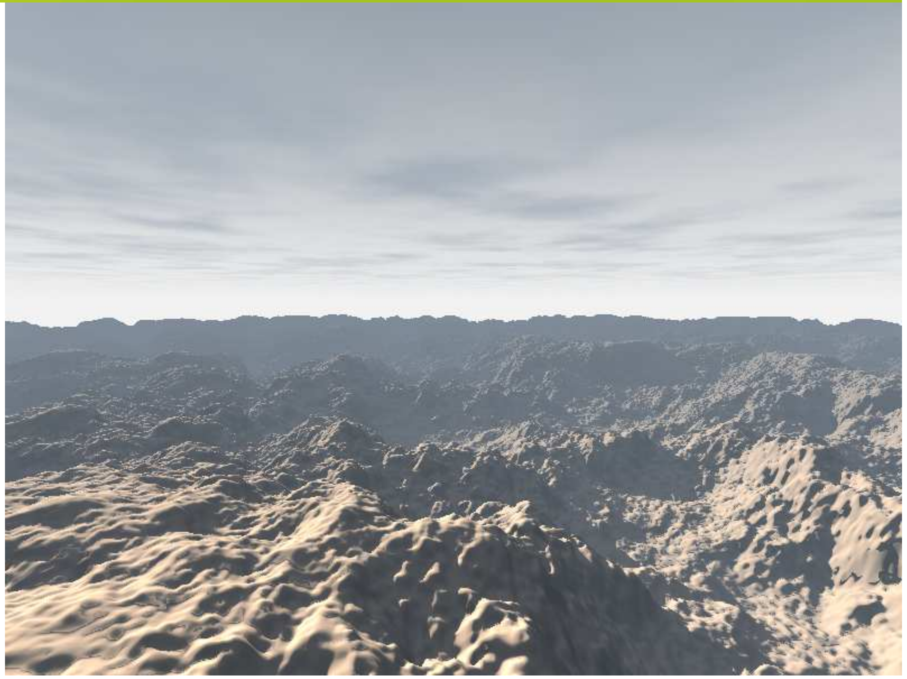
```
float map(float x, float z )
{
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f += 0.1250000f*noise2f( 4.0f*x,  4.0f*z);
    f += 0.0625000f*noise2f( 8.0f*x,  8.0f*z);
    f += 0.0312500f*noise2f(16.0f*x, 16.0f*z);
    f += 0.0156250f*noise2f(32.0f*x, 32.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;
    return f;
}
```



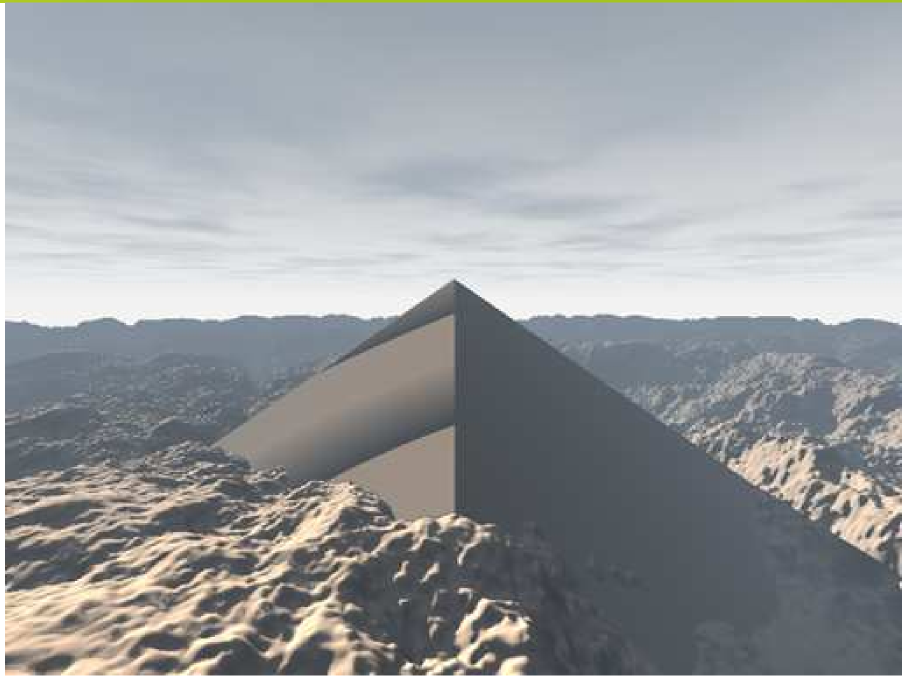
```
float map(float x, float z )
{
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f += 0.1250000f*noise2f( 4.0f*x,  4.0f*z);
    f += 0.0625000f*noise2f( 8.0f*x,  8.0f*z);
    f += 0.0312500f*noise2f(16.0f*x, 16.0f*z);
    f += 0.0156250f*noise2f(32.0f*x, 32.0f*z);
    f += 0.0078125f*noise2f(64.0f*x, 64.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;
    return f;
}
```



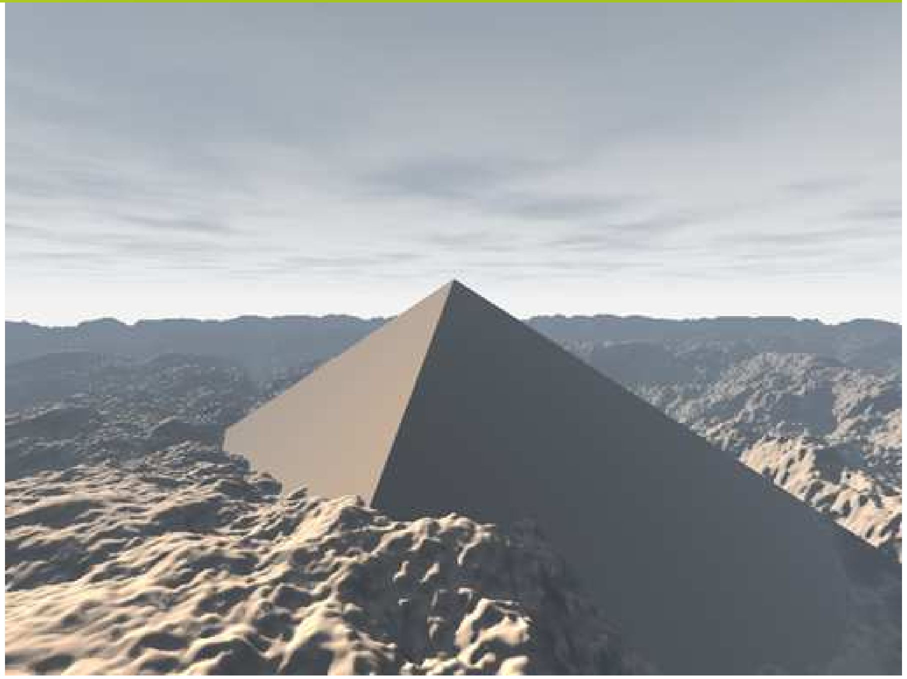
```
float map(float x, float z )
{
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f += 0.1250000f*noise2f( 4.0f*x,  4.0f*z);
    f += 0.0625000f*noise2f( 8.0f*x,  8.0f*z);
    f += 0.0312500f*noise2f(16.0f*x, 16.0f*z);
    f += 0.0156250f*noise2f(32.0f*x, 32.0f*z);
    f += 0.0078125f*noise2f(64.0f*x, 64.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;
    return f;
}}

void scattering( float *rgb, float t )
{
    // exponential decay
    float f = m2xf( -0.25f*t );
    // extintion
    rgb[0] *= f;
    rgb[1] *= f;
    rgb[2] *= f;
    // in-scattering
    rgb[0] += 0.50f*(1.0f-f);
    rgb[1] += 0.55f*(1.0f-f);
    rgb[2] += 0.60f*(1.0f-f);
}
```



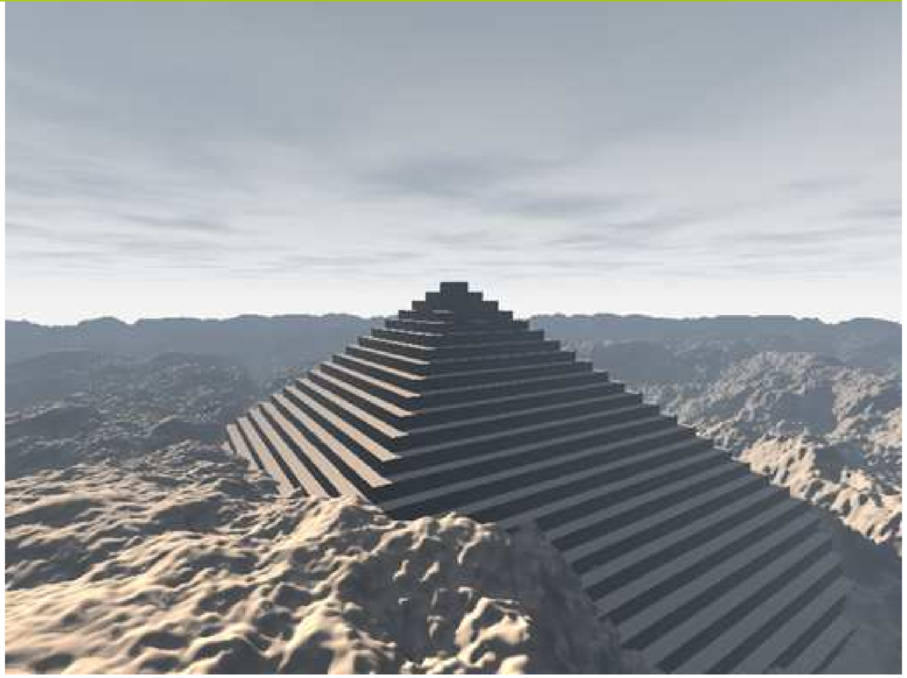
```
float map(float x, float z )
{
    // terrain
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f += 0.1250000f*noise2f( 4.0f*x,  4.0f*z);
    f += 0.0625000f*noise2f( 8.0f*x,  8.0f*z);
    f += 0.0312500f*noise2f(16.0f*x, 16.0f*z);
    f += 0.0156250f*noise2f(32.0f*x, 32.0f*z);
    f += 0.0078125f*noise2f(64.0f*x, 64.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;

    // pyramid
    float g;
    z -= 2.0f;
    g = 0.25f - fabsf(x*1.65f) - fabsf(z*1.65f);

    // select terrain or pyramid
    if( g>f ) f=g;

    return f;
}
```



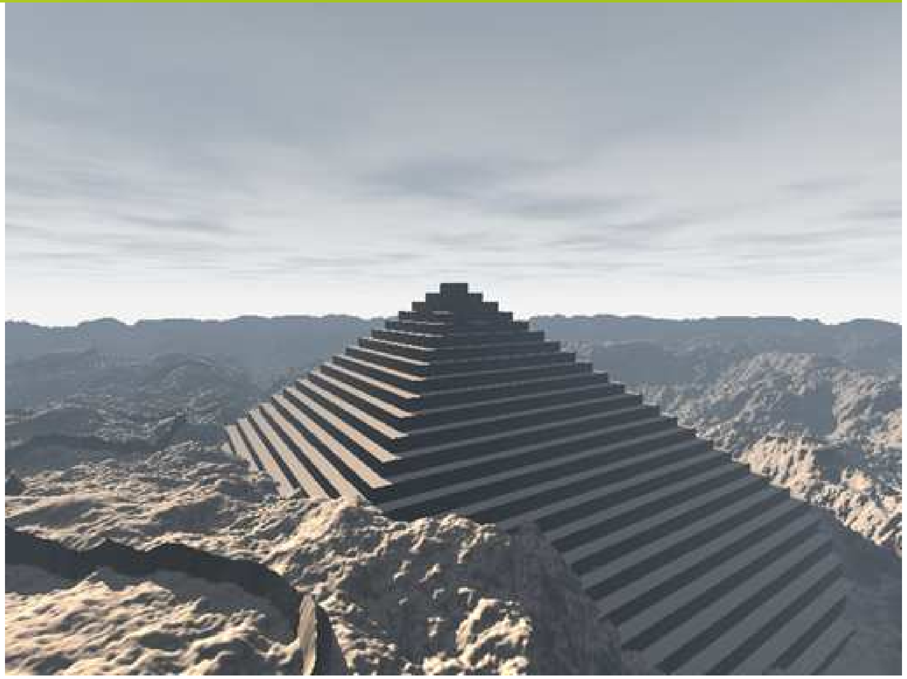
```
float map(float x, float z )
{
    // terrain
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f += 0.1250000f*noise2f( 4.0f*x,  4.0f*z);
    f += 0.0625000f*noise2f( 8.0f*x,  8.0f*z);
    f += 0.0312500f*noise2f(16.0f*x, 16.0f*z);
    f += 0.0156250f*noise2f(32.0f*x, 32.0f*z);
    f += 0.0078125f*noise2f(64.0f*x, 64.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;

    // pyramid
    float g;
    z -= 2.0f;
    float rx = x*0.9f-z*0.2f;
    float rz = x*0.2f+z*0.9f;
    g = 0.25f - fabsf(rx*1.65f) - fabsf(rz*1.65f);

    // select terrain or pyramid
    if( g>f ) f=g;

    return f;
}
```



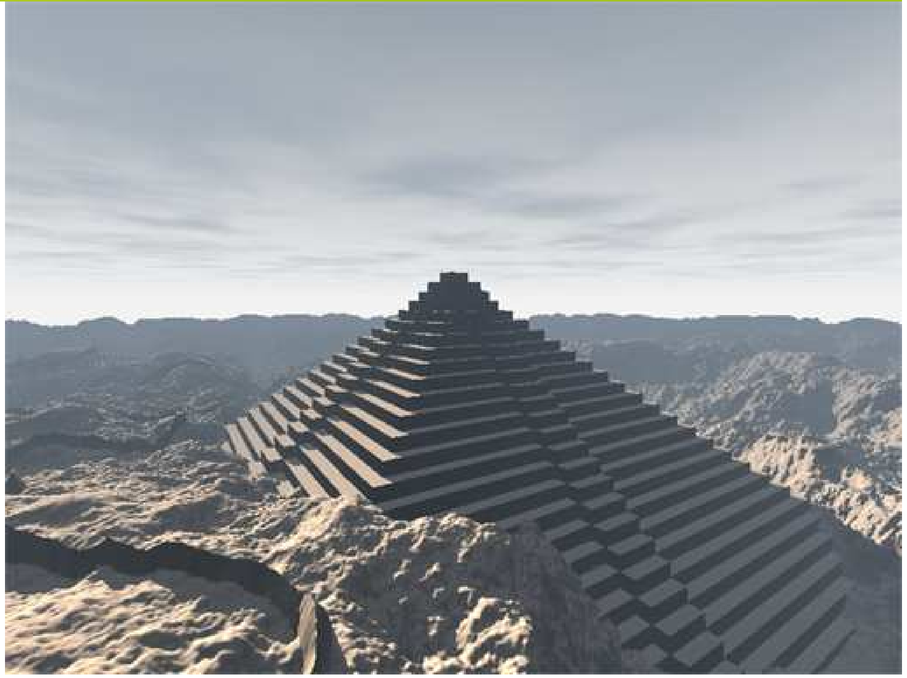
```
float map(float x, float z )
{
    // terrain
    float f;
    f  = 0.5000000f*noise2f( 1.0f*x,  1.0f*z);
    f += 0.2500000f*noise2f( 2.0f*x,  2.0f*z);
    f += 0.1250000f*noise2f( 4.0f*x,  4.0f*z);
    f += 0.0625000f*noise2f( 8.0f*x,  8.0f*z);
    f += 0.0312500f*noise2f(16.0f*x, 16.0f*z);
    f += 0.0156250f*noise2f(32.0f*x, 32.0f*z);
    f += 0.0078125f*noise2f(64.0f*x, 64.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;

    // pyramid
    float g;
    z -= 2.0f;
    float rx = x*0.9f-z*0.2f;
    float rz = x*0.2f+z*0.9f;
    g = 0.25f - fabsf(rx*1.65f) - fabsf(rz*1.65f);
    g = ((int)(g*10.0f))/10.0f;   // stairs

    // select terrain or pyramid
    if( g>f ) f=g;

    return f;
}
```



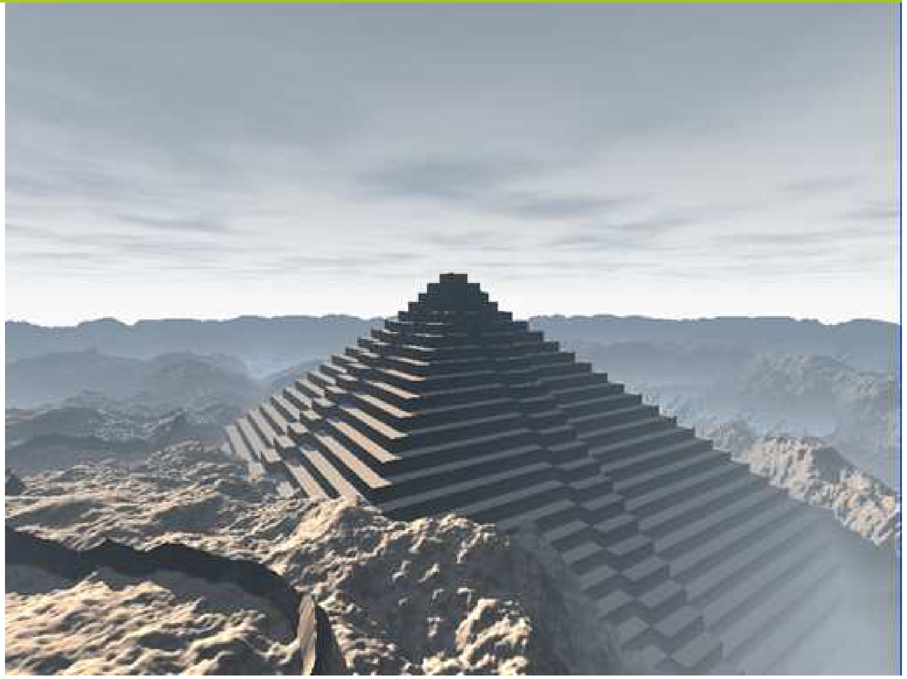
```
float map(float x, float z )
{
    // terrain
    float f;
    f  = 0.50000000f*noise2f(  1.0f*x,   1.0f*z);
    f += 0.25000000f*noise2f(  2.0f*x,   2.0f*z);
    f += 0.12500000f*noise2f(  4.0f*x,   4.0f*z);
    f += 0.06250000f*noise2f(  8.0f*x,   8.0f*z);
    f += 0.03125000f*noise2f( 16.0f*x,  16.0f*z);
    f += 0.01562500f*noise2f( 32.0f*x,  32.0f*z);
    f += 0.00781250f*noise2f( 64.0f*x,  64.0f*z);
    f += 0.00390625f*noise2f(128.0f*x, 128.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;

    // chenese wall
    float cx = x-1.5f*noise2f( 2.0f*x, 2.0f*z);
    float cz = z+1.5f*noise2f( 2.0f*x, 2.0f*z);
    float di = fabsf(cx+cz);
    if( di<0.1f )
        f += 0.1f-di;

    // pyramid
    float g;
    z -= 2.0f;
    float rx = x*0.9f-z*0.2f;
    float rz = x*0.2f+z*0.9f;
    g = 0.25f - fabsf(rx*1.65f) - fabsf(rz*1.65f);
    g = ((int)(g*10.0f))/10.0f;   // stairs

    // select terrain or pyramid
    if( g>f ) f=g;

    return f;
}
```



```
float map(float x, float z )
{
    // terrain
    float f;
    f  = 0.50000000f*noise2f(  1.0f*x,   1.0f*z);
    f += 0.25000000f*noise2f(  2.0f*x,   2.0f*z);
    f += 0.12500000f*noise2f(  4.0f*x,   4.0f*z);
    f += 0.06250000f*noise2f(  8.0f*x,   8.0f*z);
    f += 0.03125000f*noise2f( 16.0f*x,  16.0f*z);
    f += 0.01562500f*noise2f( 32.0f*x,  32.0f*z);
    f += 0.00781250f*noise2f( 64.0f*x,  64.0f*z);
    f += 0.00390625f*noise2f(128.0f*x, 128.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;

    // chenese wall
    float cx = x-1.5f*noise2f( 2.0f*x, 2.0f*z);
    float cz = z+1.5f*noise2f( 2.0f*x, 2.0f*z);
    float di = fabsf(cx+cz);
    if( di<0.1f )
        f += 0.1f-di;

    // pyramid
    float g;
    z -= 2.0f;
    float rx = x*0.9f-z*0.2f;
    float rz = x*0.2f+z*0.9f;
    g = 0.25f - fabsf(rx*1.65f) - fabsf(rz*1.65f);
    g = ((int)(g*10.0f))/10.0f;   // stairs
    if( fabsf(rx+rz)<0.1f || fabsf(rx-rz)<0.1f ) // path
        g += 0.1f;

    // select terrain or pyramid
    if( g>f ) f=g;

    return f;
}
```



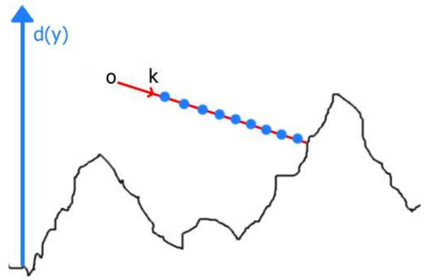
```
float map(float x, float z )
{
    // terrain
    float f;
    f  = 0.50000000f*noise2f(  1.0f*x,   1.0f*z);
    f += 0.25000000f*noise2f(  2.0f*x,   2.0f*z);
    f += 0.12500000f*noise2f(  4.0f*x,   4.0f*z);
    f += 0.06250000f*noise2f(  8.0f*x,   8.0f*z);
    f += 0.03125000f*noise2f( 16.0f*x,  16.0f*z);
    f += 0.01562500f*noise2f( 32.0f*x,  32.0f*z);
    f += 0.00781250f*noise2f( 64.0f*x,  64.0f*z);
    f += 0.00390625f*noise2f(128.0f*x, 128.0f*z);
    f = 0.5f+0.5f*f;
    f = f*f*(3.0f-2.0f*f);
    f = f*f*(3.0f-2.0f*f);
    f = -2.5f + 1.5f*f;

    // chenese wall
    float cx = x-1.5f*noise2f( 2.0f*x, 2.0f*z);
    float cz = z+1.5f*noise2f( 2.0f*x, 2.0f*z);
    float di = fabsf(cx+cz);
    if( di<0.1f )
        f += 0.1f-di;

    // pyramid
    float g;
    z -= 2.0f;
    float rx = x*0.9f-z*0.2f;
    float rz = x*0.2f+z*0.9f;
    g = 0.25f - fabsf(rx*1.65f) - fabsf(rz*1.65f);
    g = ((int)(g*10.0f))/10.0f;    // stairs
    if( fabsf(rx+rz)<0.1f || fabsf(rx-rz)<0.1f ) // path
        g += 0.1f;

    // select terrain or pyramid
    if( g>f ) f=g;

    return f;
}}

void fog( float *rgb, float t, const float *rayDir)
{
    float fog = (1.0f-m2xf(-2.5f*rayDir[1]*t) ) / rayDir[1];
    collerp(rgb, fog*0.01f, rgb, fogColor);
}
```



- Density of air-atmosphere decays with altitude $d(y) = a \cdot e^{-by}$
- A ray with origin ***o*** and direction ***k*** is $r(t) = o_y + t \cdot k_y$
- Accumulated air density:

$$D = \int_0^T d(y(t)) \cdot dt$$

$$D = \int_0^T d(o_y + t \cdot k_y) \cdot dt \quad = \quad a \cdot e^{-b \cdot o_y} \frac{1 - e^{-b \cdot k_y \cdot T}}{b \cdot k_y}$$



Index

3D procedural drawing

• 2.5D is easy to implement, fast to render, and easy to wirk with. But it has a big drawback, the 3D is just a heighmap, and so it cannot draw a concave scene.

• In Ixaleno the terrain and alien bases and road where on the heighmap, and the spaceships where done differently because of this limitation of heighmaps.

• 3D is as easy to use as 2.5D (as procedural artists), and shares all the advantajes of 2.5D.

• If done cleverly, it is not slower to render than 2.5 maps ;)

• The idea is to basically write a map of (x, y, z) that defines for each point in space the presence of an object.

There was an interesting procedural full 3d image in this slide.
After some explanations on it, speech attendees were flashed
and their short term visual memory erased.

work-in-progress by rgba

Index

Conclusions

• 4k graphics are an extremelly interesting competition, a good platform for experimentation on rendering techniques, and to show your little artist inside.

• Image compression is not that much of an option (yet).

• Proceduralism is fun to play with. Simple geometric concepts plus perlin noise and smoothstep to the powa.

• In 3D things become “easier” and more impressive.

inspire!
inspire-demoscene.org
9-11/05/2008
errenteria/spain

“elexiane“, 4k executable graphic by rgba