Hardware pro počítačovou grafiku
NPGR019

Hardware graphics effects

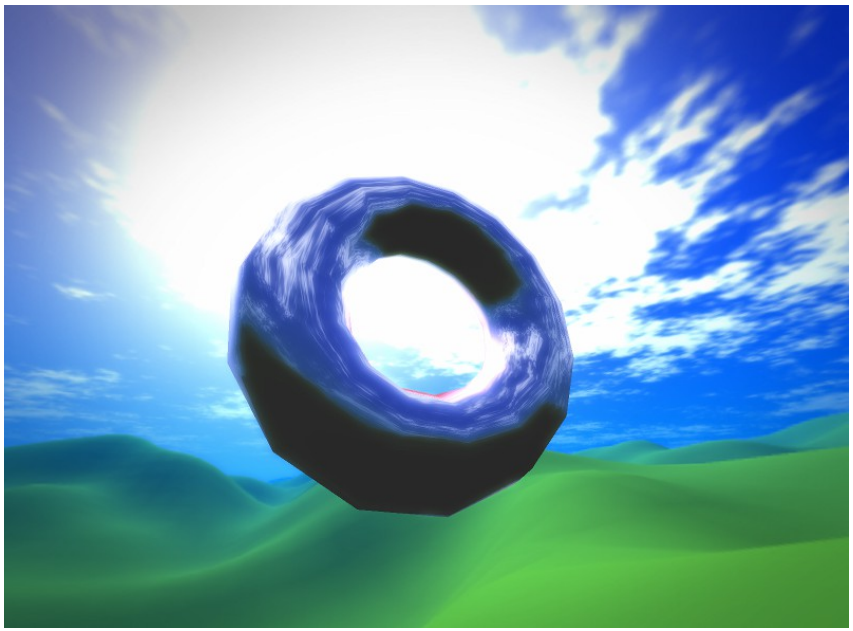
Josef Pelikán
Jan Horáček

http://cgg.mff.cuni.cz/
MFF UK Praha

2012

Table of contents

Bloom effect



Lowpass Image Filtering

- Some fullscreen effects need to blur the image
- In HDR rendering used to emphasize overexposed parts
- "Cheap" effect to:
  1. Increase realism (photo-like appearance)
  2. Change mood of scene (dreamy-like appearance)
- Often used as a means of calling a game *Next-Gen*

Box filter

- Image subsampling
- Simplest technique, has support on most current and even older HW
- Needs only rendering to buffer

Implementation

1. Render scene to offscreen buffer (texture)
2. Buffer for next step rendering half-size than current image
3. Draw fullscreen quad with texture coordinates so that pixels are placed inbetween current texels
   - This performs linear interpolation of texels "for free" (in texturing unit of graphics card)
4. Go to step 2 until sufficient resolution
5. Map low-resolution result on fullscreen quad and alpha-blend with scene

Properties

- Advantages
  - Very fast
  - 2-4 iterations enough for strong effect

Properties

- Advantages
  - Very fast
  - 2-4 iterations enough for strong effect
- Disadvantages
  - Resolution-dependent
  - Linear filtering artifacts

Gauss filter

- Overcomes the problem with linear filtering artifacts by performing convolution with Gauss kernel
- Does not suffer by linear artifacts
- Gauss filter separable: 2D convolution == 2x1D convolution

Implementation

- Use row from *Pacal's Triangle* as convolution kernel
  - Caution: different rows have different σ

$$
\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1
\end{array}
$$

Implementation

1. Downsample to ease computational burden
2. Perform convolution in X direction
3. Perform convolution in Y direction
4. Map filtered result on fullscreen quad and alpha-blend with scene

CPU vs. GPU Performance Example

- Filtering 4096x4096xRGB image
- "Stupid" implementation in C
  - Simple *for* cycle, contains *modulo* for each pixel
  - 7425ms

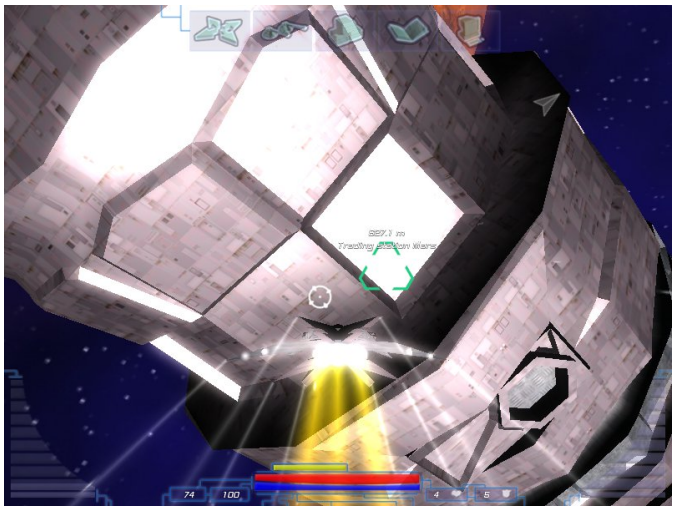
CPU vs. GPU Performance Example

- Filtering 4096x4096xRGB image
- "Stupid" implementation in C
  - Simple *for* cycle, contains *modulo* for each pixel
  - 7425ms
- Optimized implementation in IA32 assembler
  - Special cases handled separately, around 600 instructions
  - 1560ms

CPU vs. GPU Performance Example

- Filtering 4096x4096xRGB image
- "Stupid" implementation in C
  - Simple *for* cycle, contains *modulo* for each pixel
  - 7425ms
- Optimized implementation in IA32 assembler
  - Special cases handled separately, around 600 instructions
  - 1560ms
- OpenGL ARB_fragment_program
  - Around 30 instructions
  - $<<$ 15ms

Examples



Signum

TES IV:Oblivion

Examples

Halo 3

NFS: Most Wanted

Notes

- Render in full resolution and then downsample/filter
  - Otherwise temporal artifacts
- Use alpha channel on non-HDR textures for bloom

Ambient Occlusion

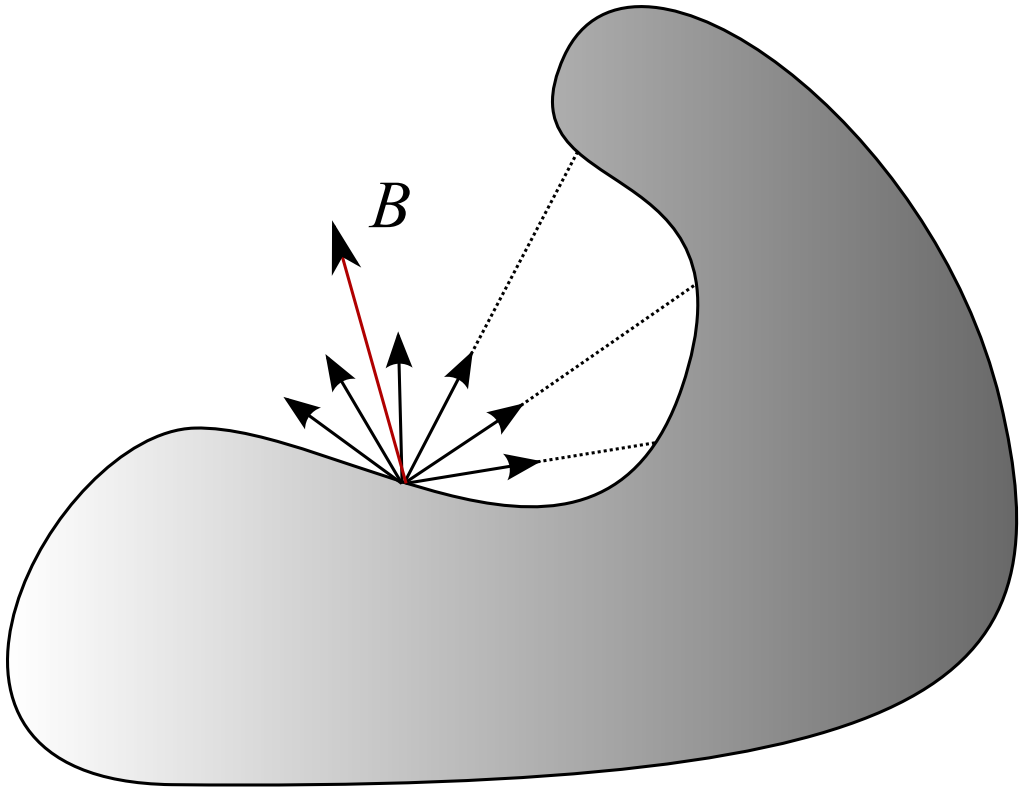
Goal of Ambient Occlusion

- Simulate ambient light coming from all directions
- Take into consideration accessibility of polygon/point
- Simple point light rendering creates unrealistic "hard" appearance
- Inner corners and holes appear darker

Raytraced version

- From each point trace rays into surrounding
- Discard rays that hit the object itself
- Rays that do not hit any other polygon from the object sample an environment map
- Sum samples and use its average as incoming ambient light

Raytraced version



Preprocessing for realtime display

- Preprocess with raytraced version
  1. From each point trace rays into surrounding
  2. Store unoccluded/total rays ratio F
  3. Store average direction of unoccluded rays B
- With good tesselation - enough per poly
- Coarse tesselation - use texture maps

Rendering on older HW

- Use unoccluded/total rays ratio F for modulating color texture (like lightmap)
- Applicable as single pass even on HW like 3Dfx Voodoo2
- Adds a sort of "visual depth" to the objects
- Can be simply combined with many other effects

HW Preprocessing

- Use array of lights surrounding the object (100-1000)
- For each light create shadow map
- Combine shadow maps during rendering
  - Many textures = multiple passes
- Need high precision accumulation buffer (16bit is not enough)
- Computationally intensive, but still much faster than SW raytracing

HW Preprocessing 2

- Combining in camera space is good for single image display, but not for realtime performance
- Use vertex program for unwrapping model according to the texture coordinates, other parameters stay the same
- Render result into the texture instead of screen space
- Use texture for realtime rendering

Environmental light on modern HW

- Environmental map in lat-long projection (mipmap friendly)
- Precompute maps of B and F (offline, but only once for a model)
- During rendering use B as sample direction into the environment map
- Estimate sample area from F
- Use fast HW mipmapping for sample area averaging

Environmental light on modern HW 2

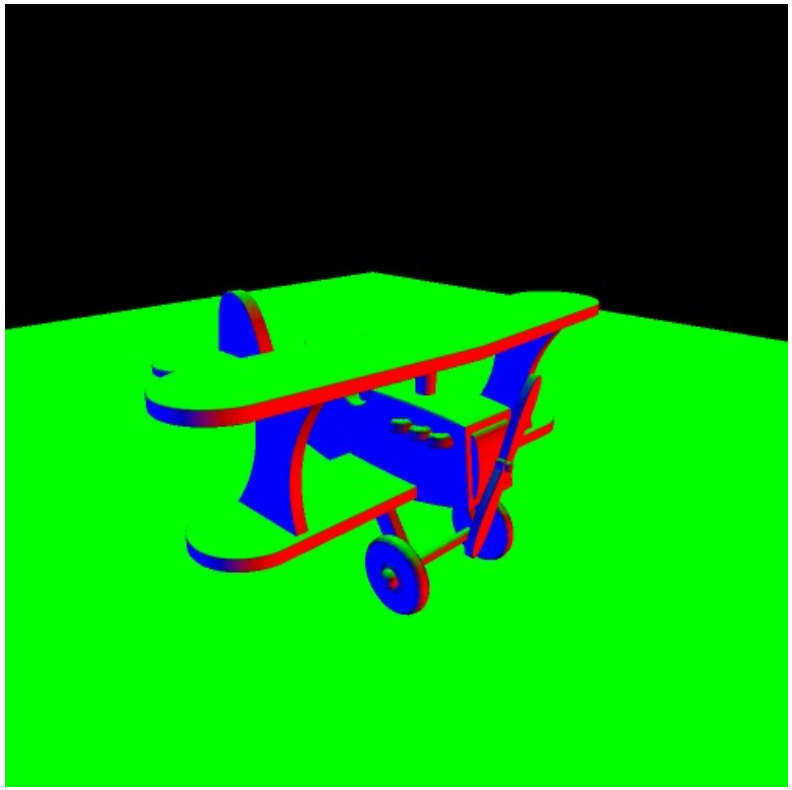
- Physically incorrect, but gives visually good results, incorrect parts usually covered
- Applicable on animated models - compute for keyframes and interpolate B and F

Examples - Accessibility

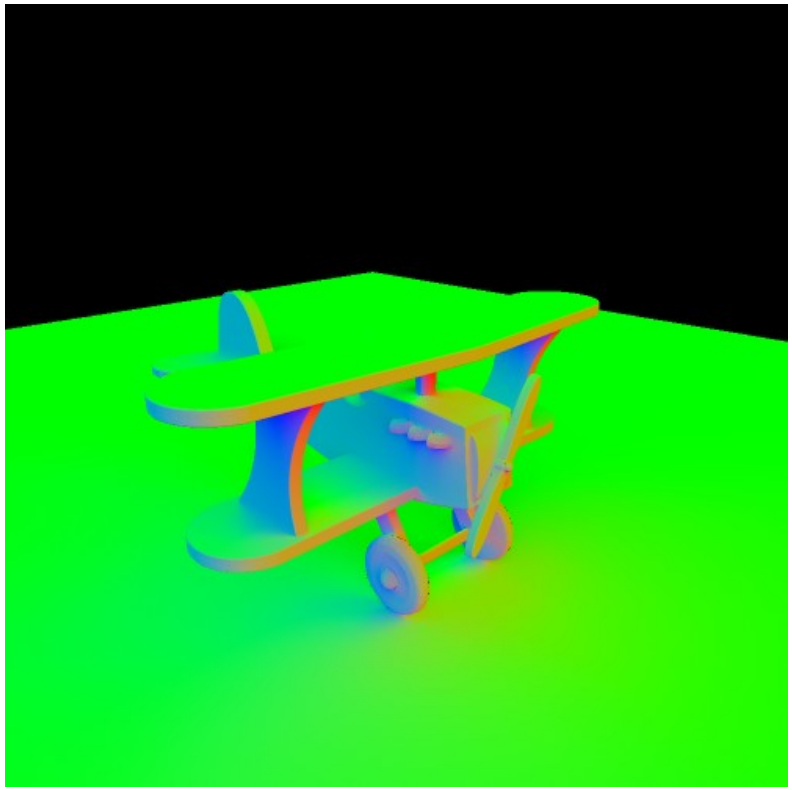
Phong shading

Accessibility coefficient

Examples - Normals



Model normals



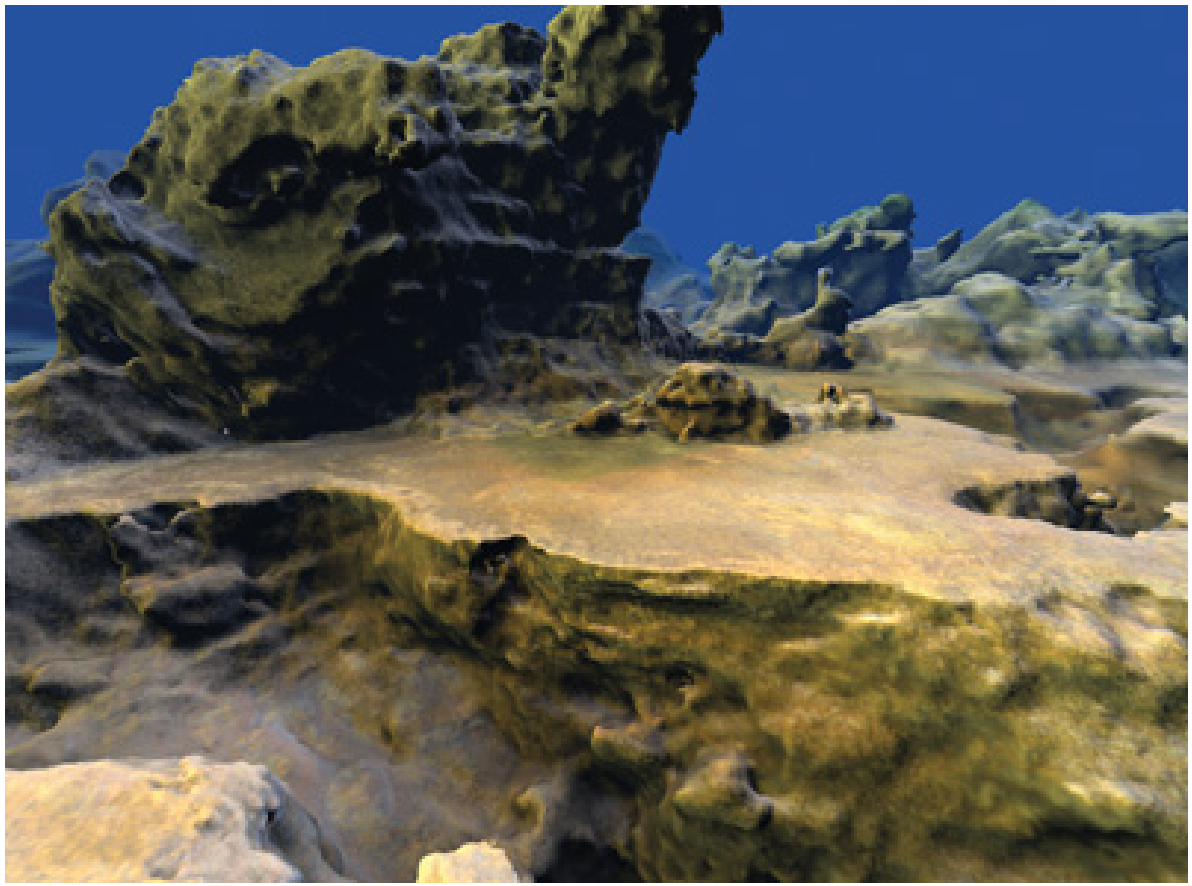
Average unoccluded ray B

Examples - Comparison of results

Phong shading

Environment lighting

Terrain rendering



Terrain Rendering Approaches

- Heightmaps
  - Do not handle well complex geometry such as overhangs, caves, etc.
- Handcrafted
  - Too painful work for more complex terrains
  - Area limited

Terrain as a 3D function

- Inspired by volumetric data rendering (medicine, biology, etc.)
- Function $f(x, y, z) = \text{density}$
- Terrain generated as *isosurface* at $f(x, y, z) = 0$

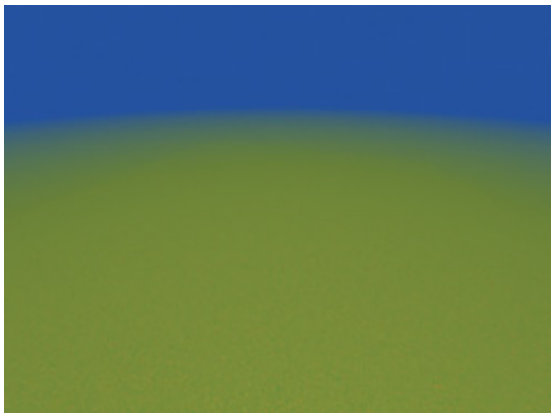
Examples

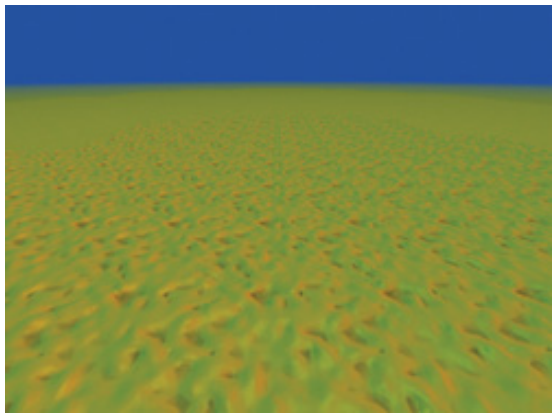
Flat land: $f(x, y, z) = y$
Planet: $f(x, y, z) = \sqrt{(x^2 + y^2 + z^2)}$

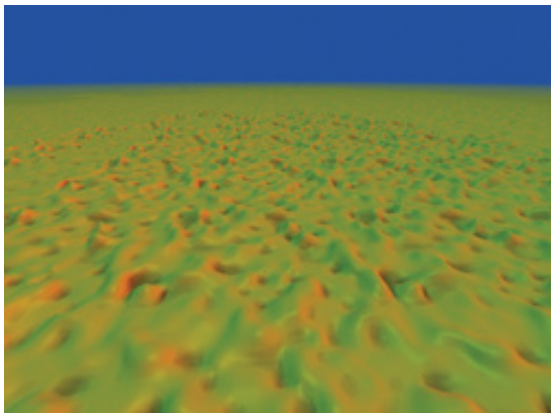
Noise Function

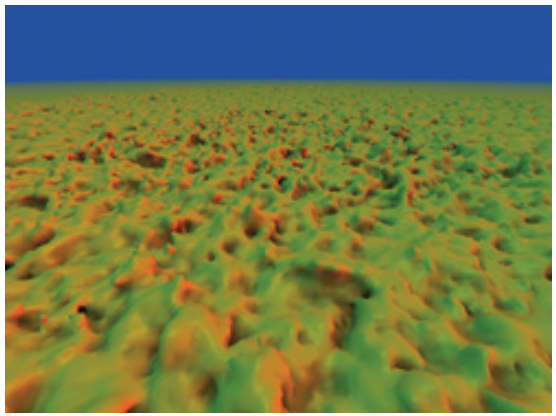
- Change density function with noise
- Noise generated in 3D texture
- Different amplitudes and octaves generate different features
- $density(x, y, z) + Am \cdot noise(Oc \cdot x, Oc \cdot y, Oc \cdot z)$
- Usually 8 octaves enough for creating terrain with all kinds of features from the size of hills to the size of small crevices

Examples - Simple terrains







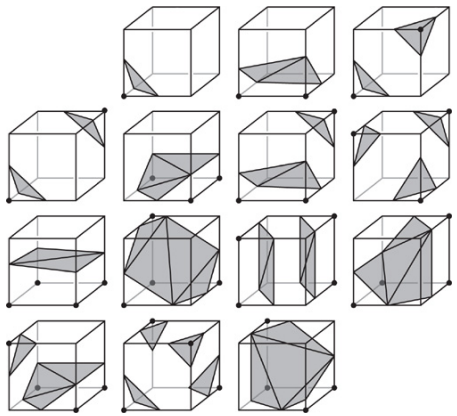


Creating Polygons

1. Divide world into blocks and generate & display only visible blocks of 32^3 cubes
2. Sample density function and store it in 3D texture (a set of 2D textures)
3. Use *Marching cubes* algorithm for creating polygonal surface in geometry shader
4. Calculate lighting and textures/texture coordinates
5. Discard invisible blocks

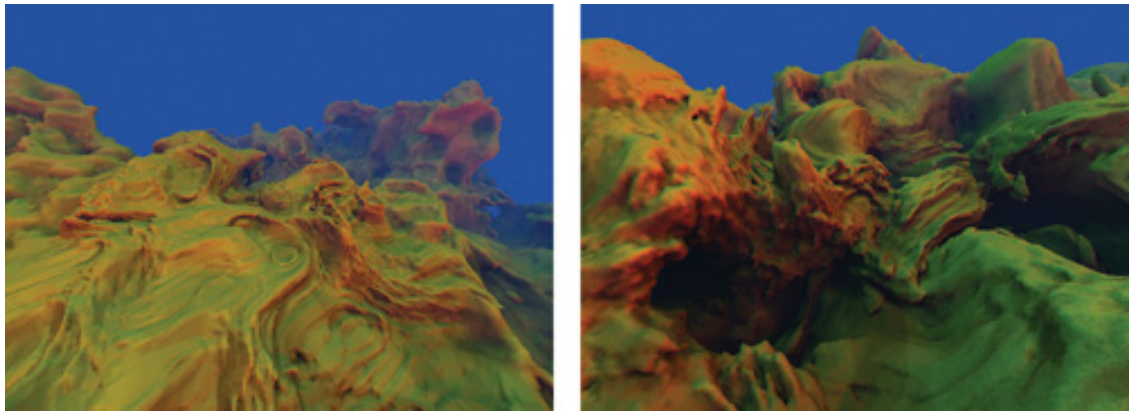
Algorithm improvements

- Geometry shader can generate only indices, move work to vertex shader for generating actual parameters
- Share vertices between neighbouring cubes



Additional Effects

- Warp world space coordinates to create organic-like effects



Warped world space coordinates

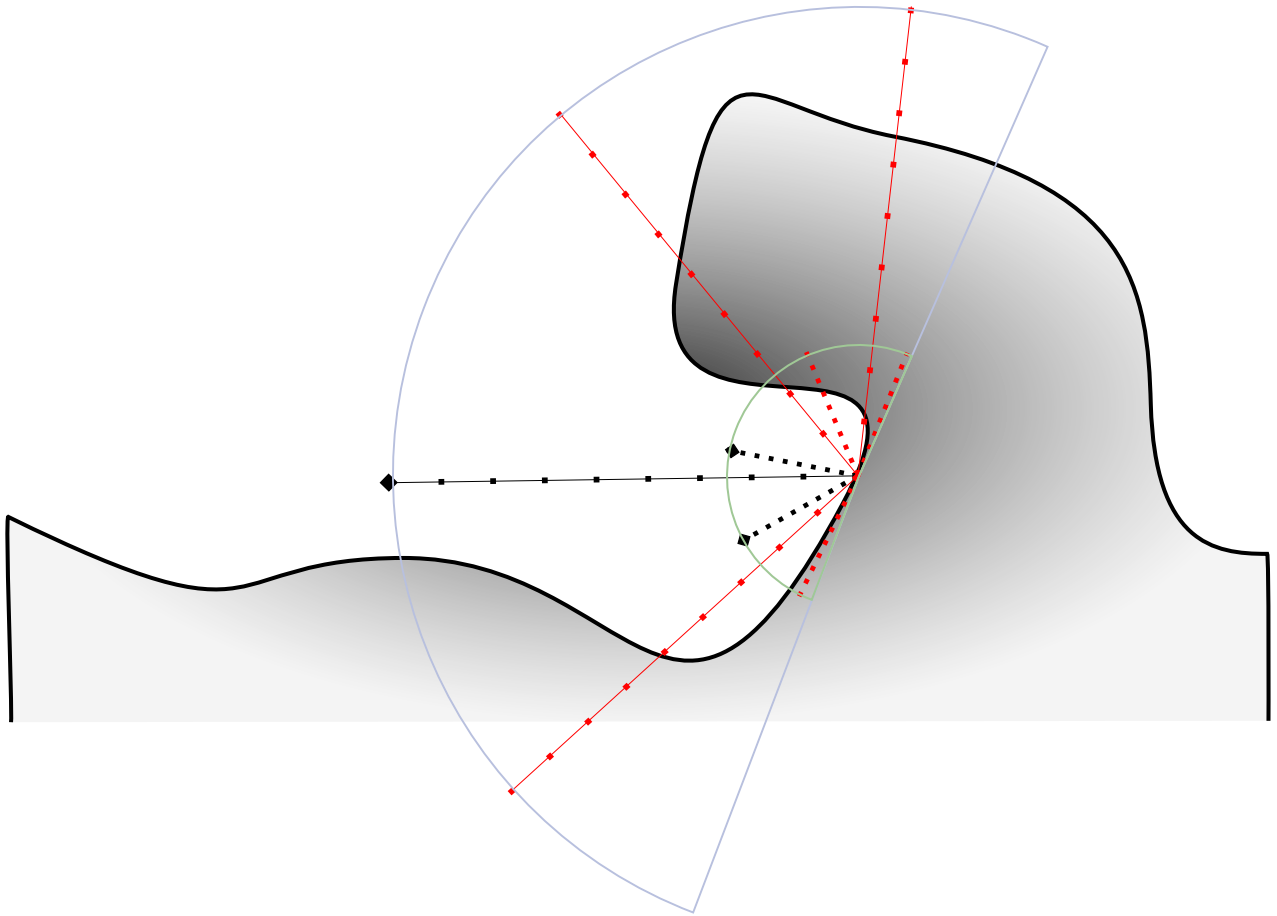
User Control

- User data for manual control of various parameters can be easily used
  - For example landing pads for helicopters, roads, flat lands, ...
- Textures for local control of noise behaviour
  - Amount of noise, octaves/amplitudes, flat lands
  - Rocky area, organic-like area, ...

Lighting

- Ambient occlusion for terrain shading
- Sample *density volume* by many samples in close neighborhood of polygon for small crevices shading
- Sample broader part of *density function* by fewer samples to shadow big caves/valleys
  - May reach outside the volume - sample directly function instead of precomputed volume
- Use *soft decision* version for smoother result
  - *Hard decision* creates shading artifacts

Lighting Schema



Sampling point neighborhood

Dynamic objects

- Density function may be used for other computation, such as collision detection
  - Density behaves *similarly* to distance function
- Ambient light - sampling local neighborhood of object

Literature

- GPU Gems 1,2,3
- http://www.gamedev.net
- http://www.fusionindustries.com