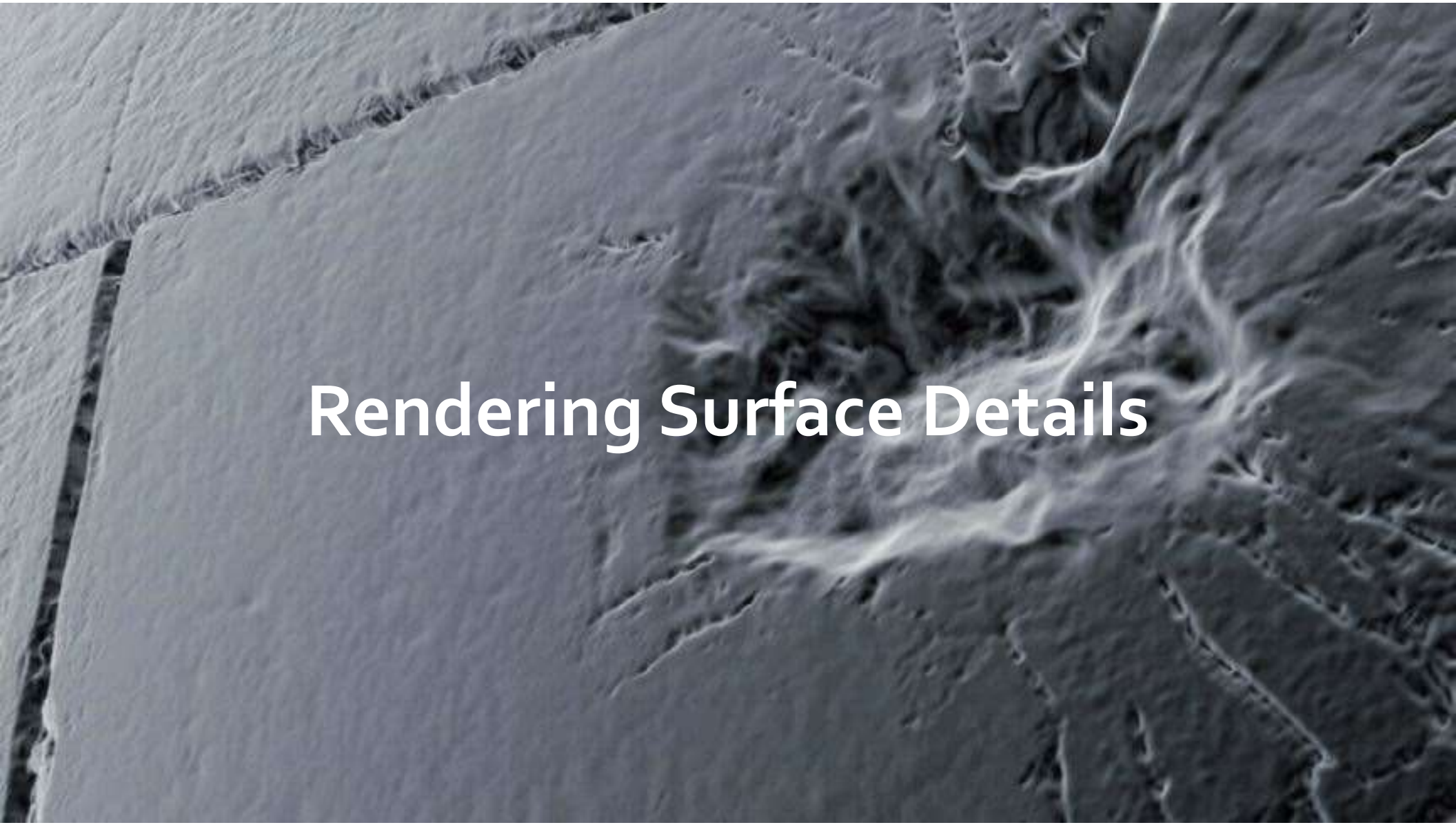
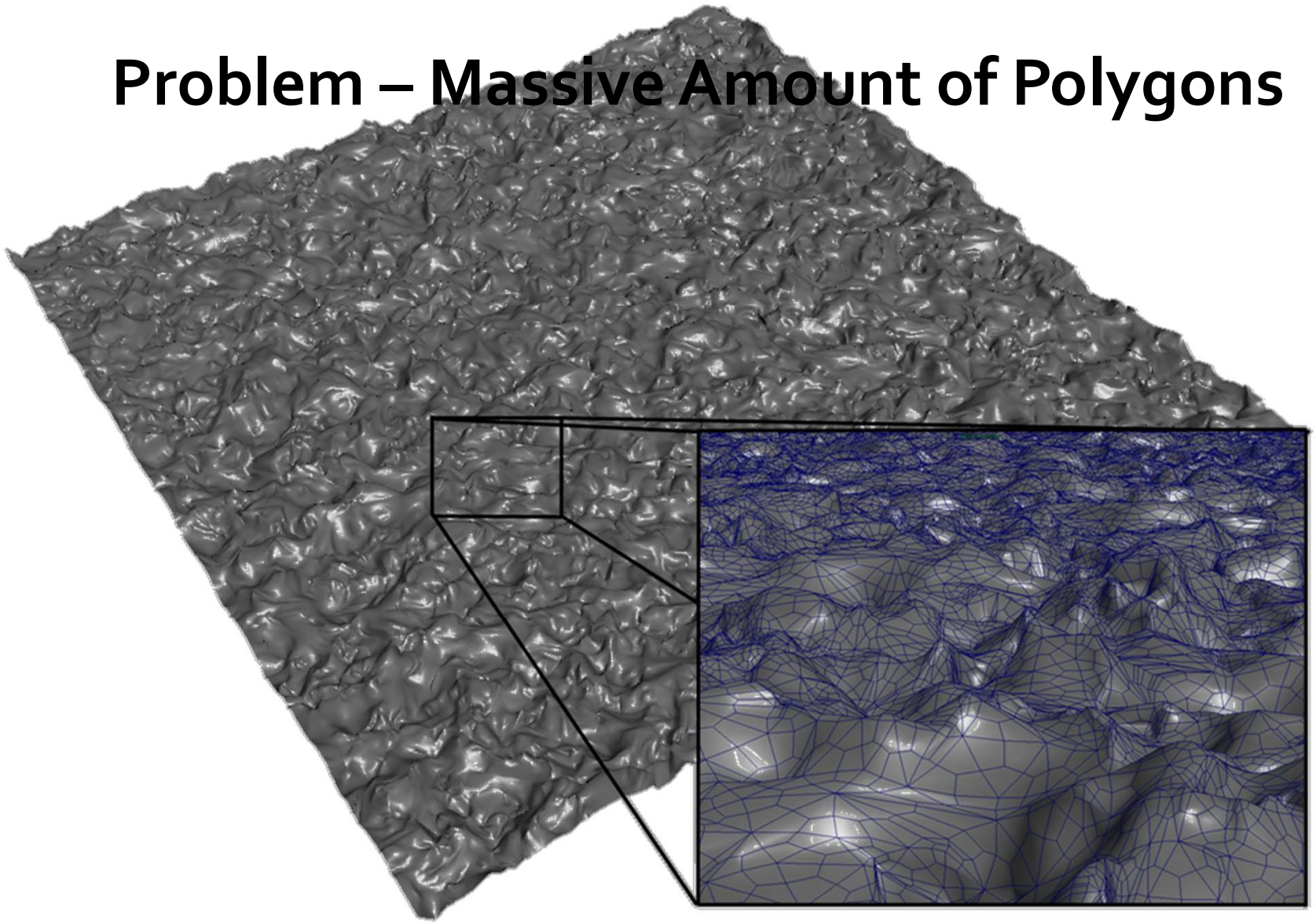

Rendering Surface Details

Problem – Massive Amount of Polygons



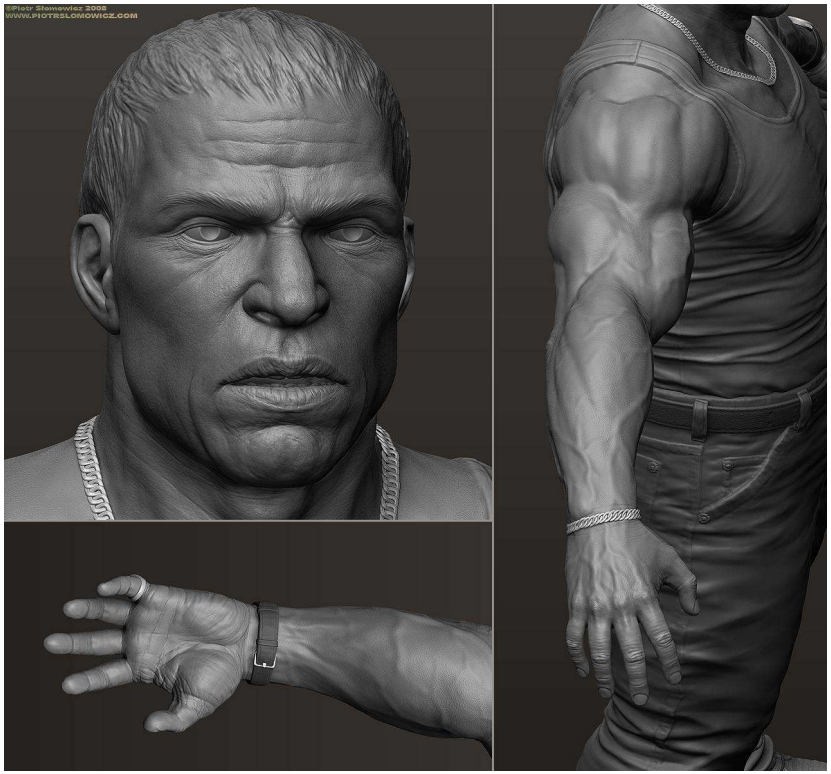
Idea

- Highly detailed surface look without huge models

+

Detail information

=



Approaches

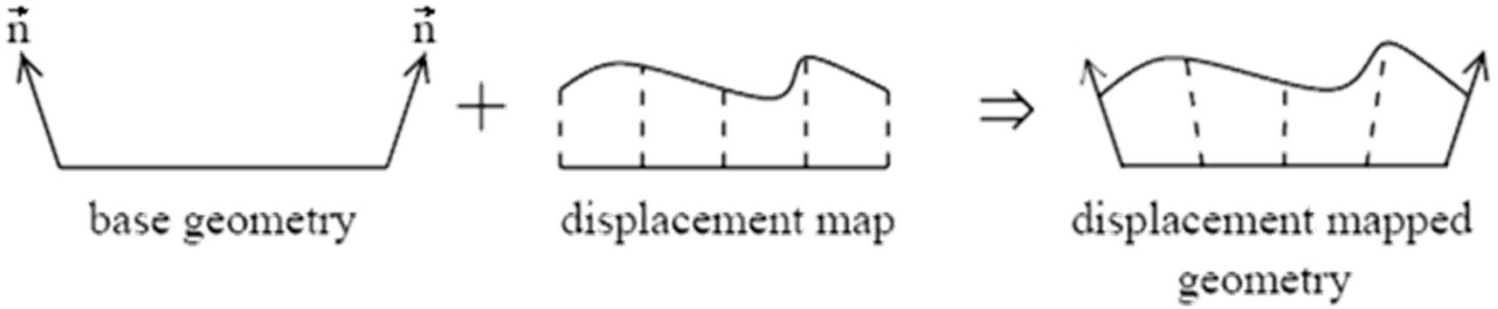
- Geometric
  - Displacement Mapping
  - Volume slice rendering
  - Ray tracing
- Image Space
  - Normal mapping
  - Parallax mapping
  - Relief textures
  - BRDFs (Bidirectional Reflectance Distribution Functions)

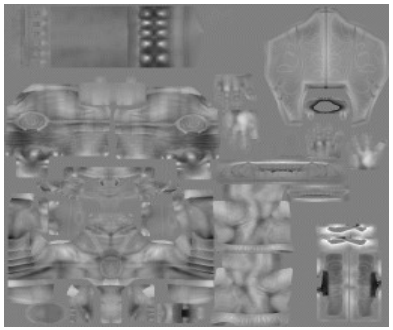
Rendering Surface Details

Displacement Mapping

Displacement Mapping

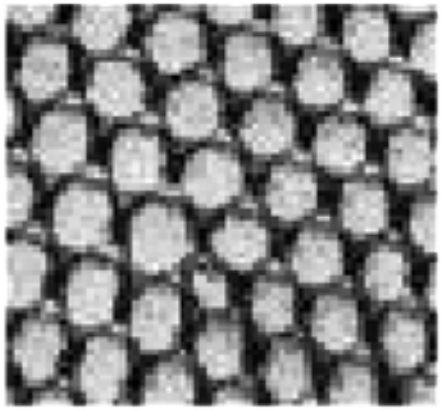
- A displacement map specifies displacement in the direction of the surface normal, for each point on a surface

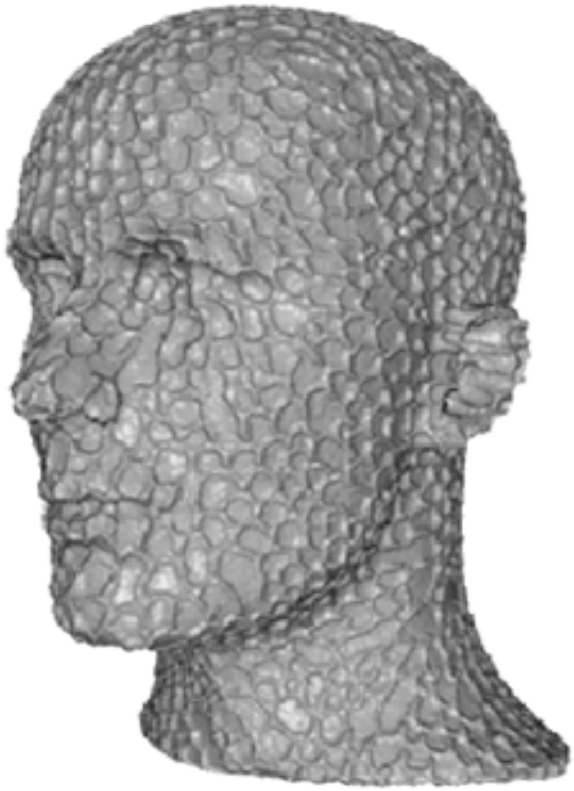




Displacement Mapping – Idea

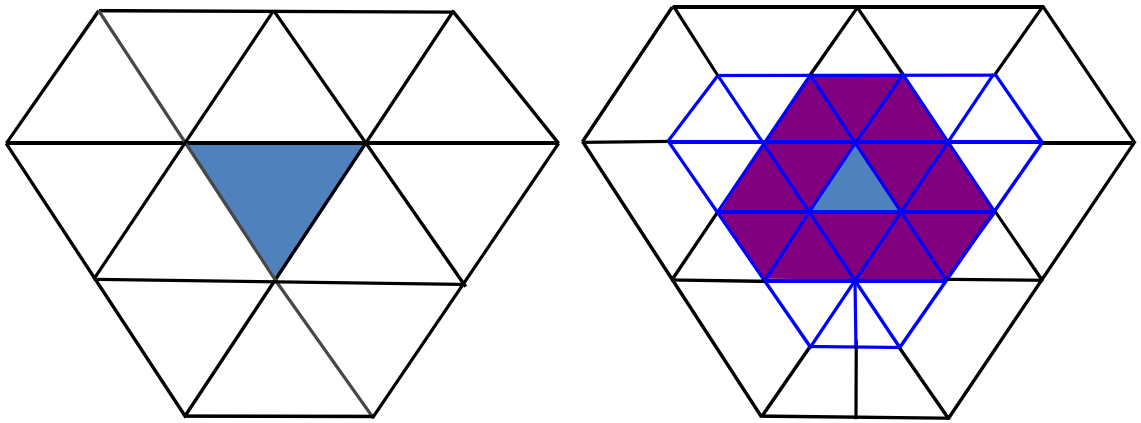
- Shift all points on the surface along their normal vectors
- p' = p + displacementTex(p) * n
- How can we render a model given as a set of polygons, and a displacement map?





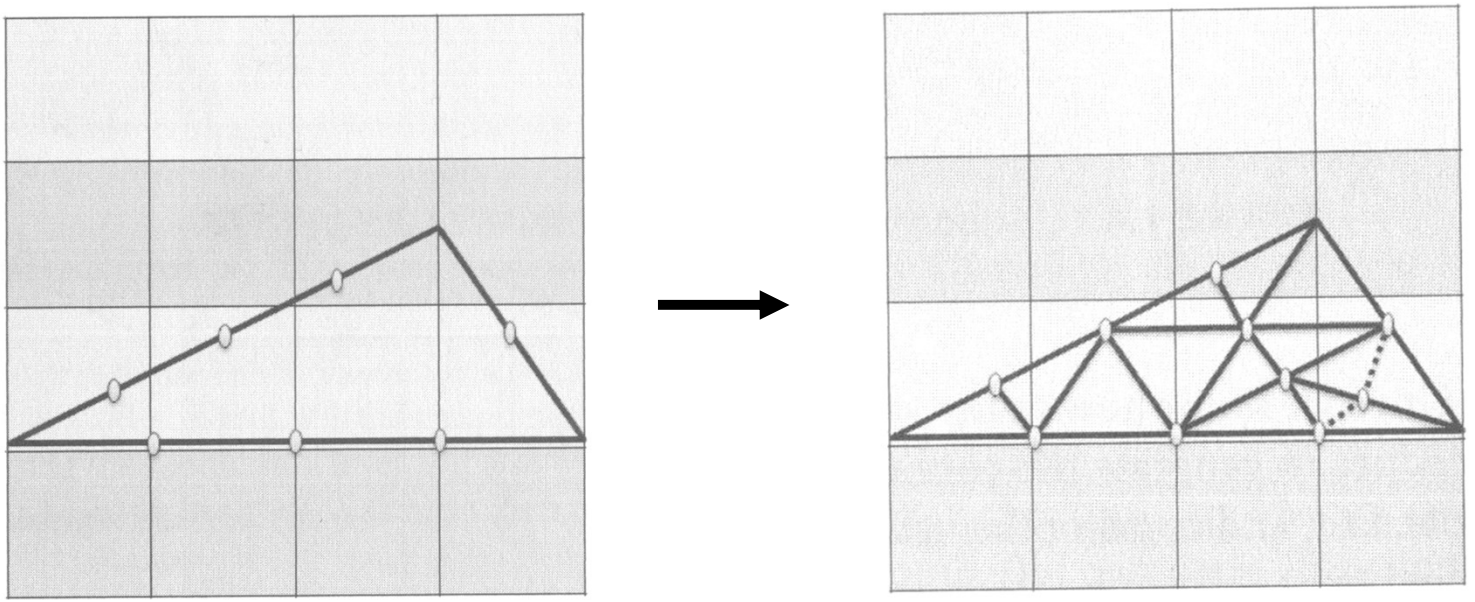
Subdivide and Displace

- Subdivide each polygon, till?
- Displace each vertex along normal using displacement map
- Many new vertices and triangles
- Improvements
  - Adaptive subdivision



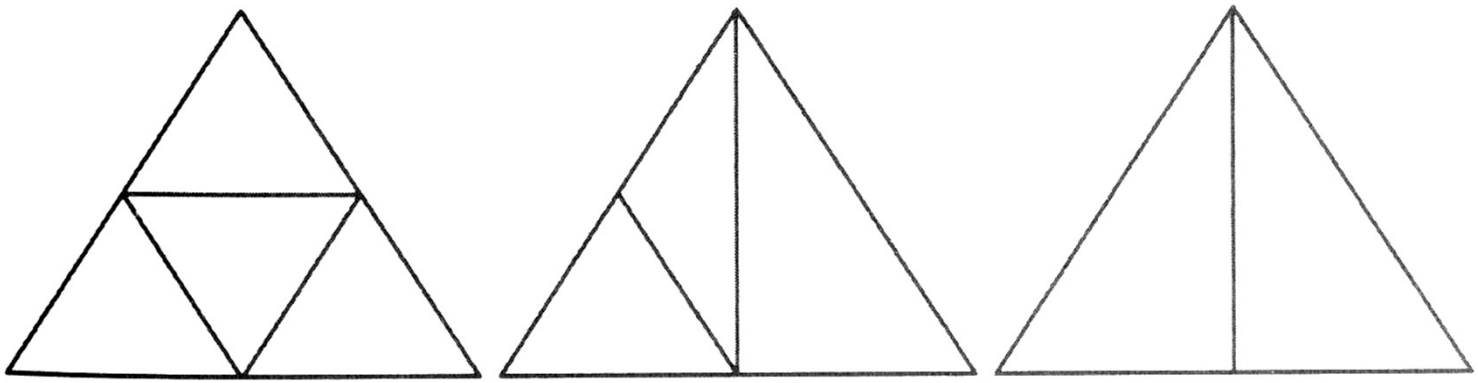
Simple Adaptive Subdivision

- Idea: subdivision based on edge length
- At least one triangle per pixel
- Efficient?



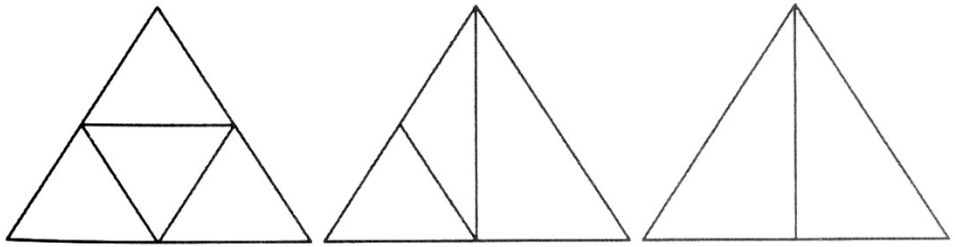
Simple Adaptive Subdivision

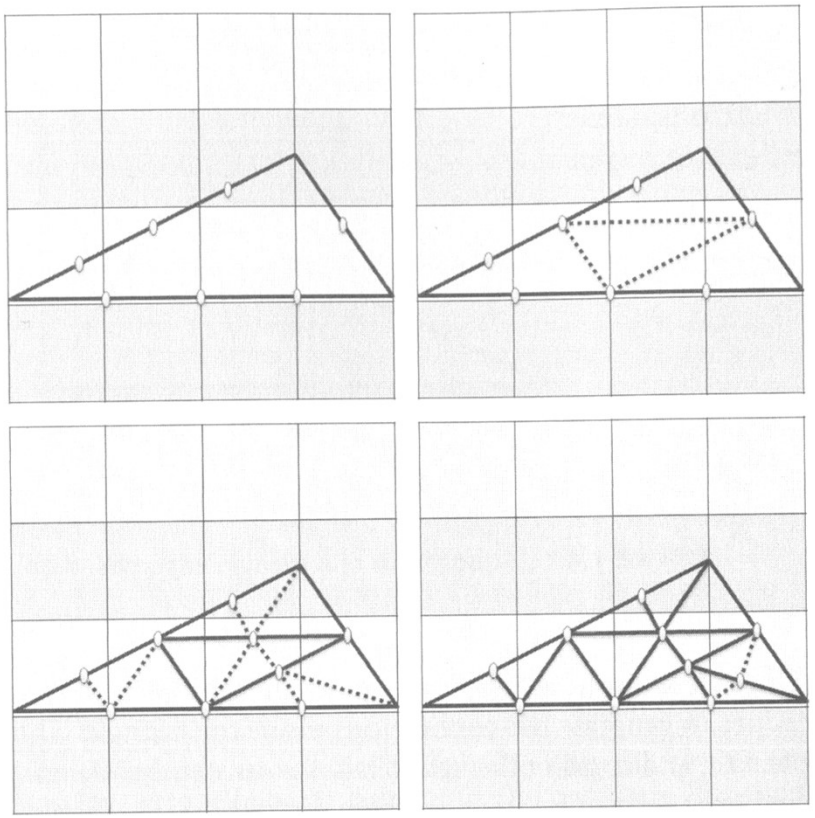
- Pre-computed tessellation patterns
  - 7 possible patterns
  - 3 if we do rotation in code



Simple Adaptive Subdivision

- Precomputed tessellation patterns
- Recursive subdivision





Rendering Surface Details

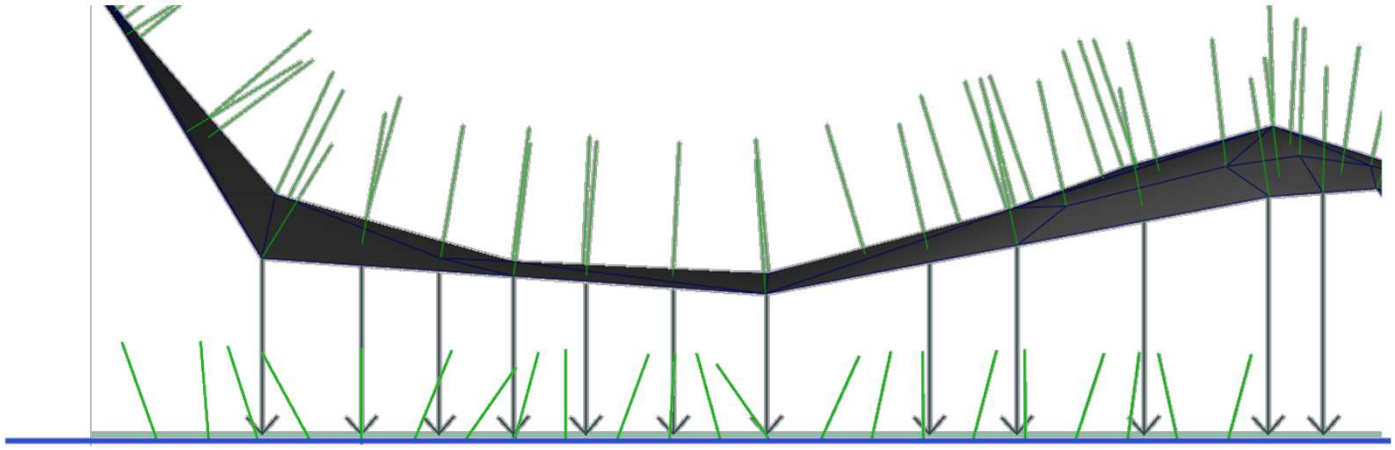
Normal(Bump) Mapping

Normal Mapping

- Bump/normal mapping invented by Blinn 1978.
- Efficient rendering of structured surfaces
- Enormous visual Improvement without additional geometry
- Is a local method
  - Does not know anything about surrounding except lights
- Heavily used method
- Realistic AAA games normal map every surface

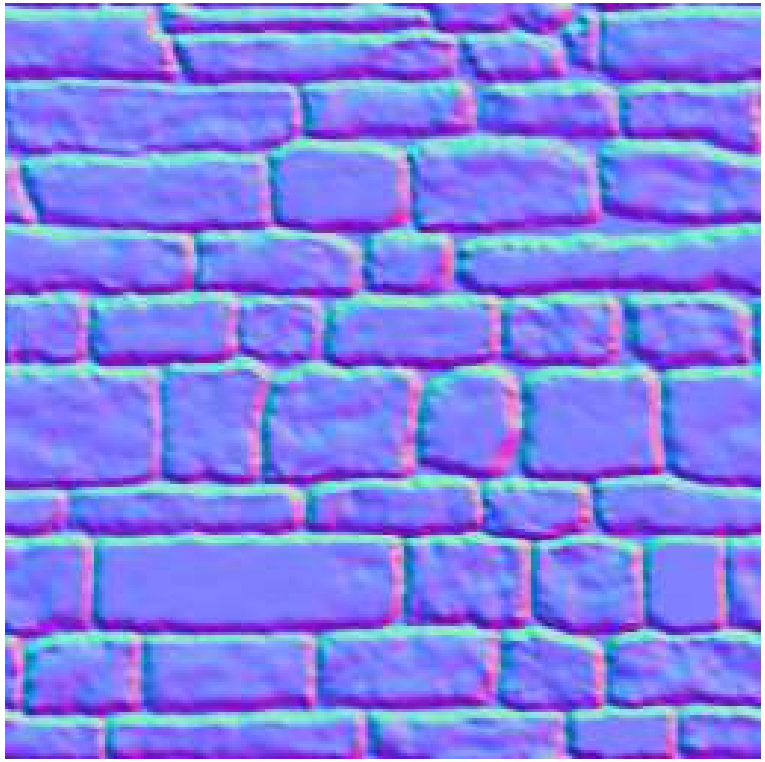
Normal Mapping

- Idea: illumination is not directly dependent on position
- Position can be approximated by carrier geometry
- Idea: transfer normal to carrier geometry



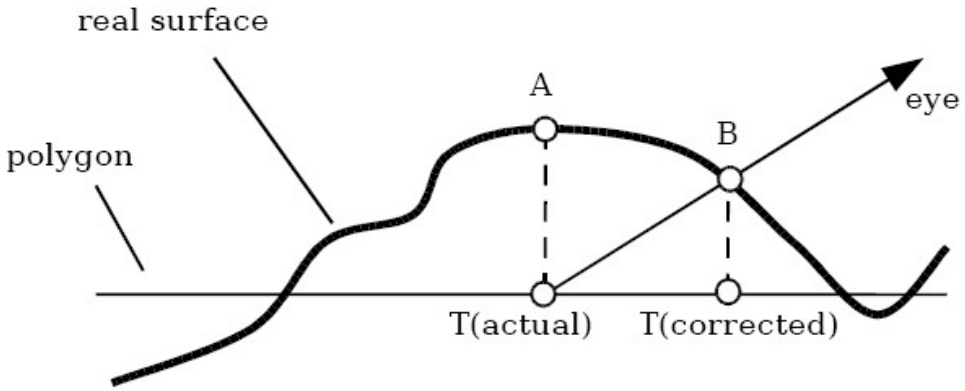
Normal Mapping

- Result: Texture that contains the normals as vectors
  - Red X
  - Green Y
  - Blue Z
  - Saved as range compressed bitmap ([-1..1] mapped to [0..1])
- Directions instead of polygons!



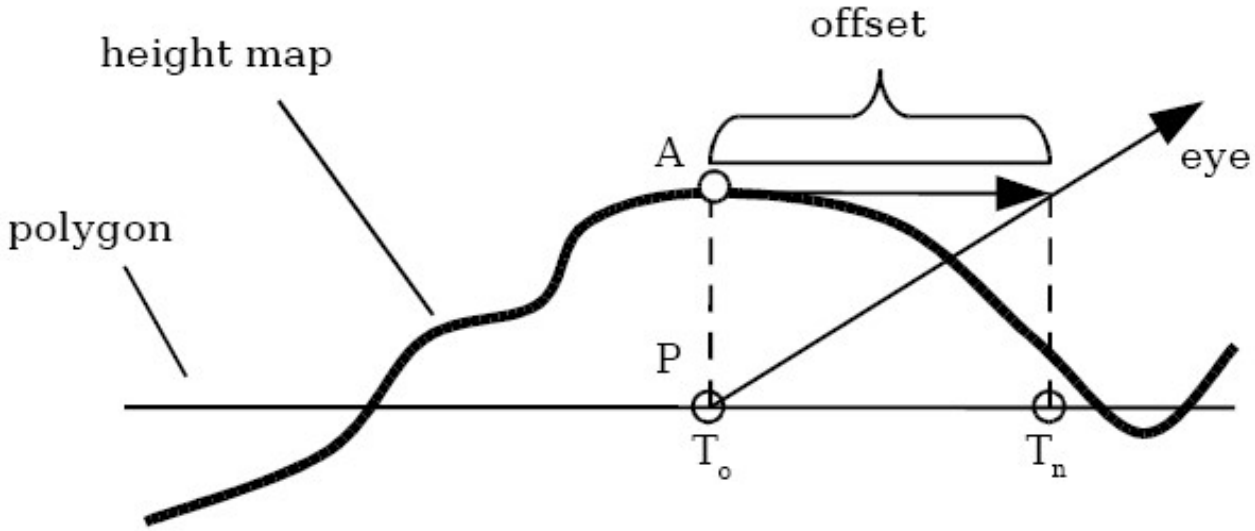
Parallax-Normal Mapping

- Normal mapping does not use the height field
  - No parallax effect, surface is still flattened
- Idea: distort texture lookup according to view vector and height field
  - Good approximation of original geometry



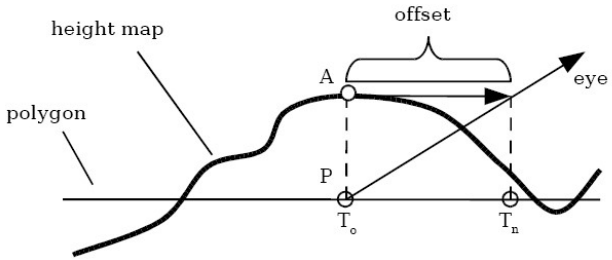
Parallax-Normal Mapping

- We want to calculate the offset to lookup color and normals from the corrected position Tn to do shading there



Parallax-Normal Mapping

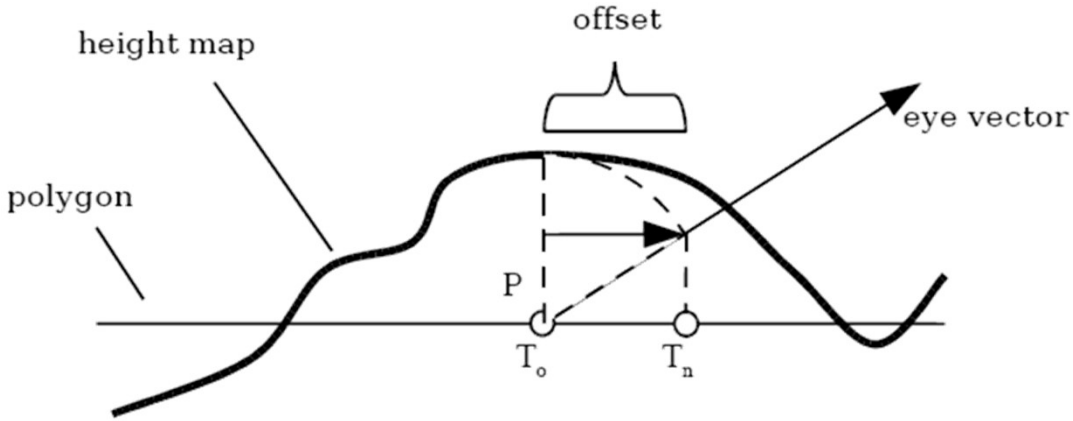
- Rescale height map h to appropriate values:
  hn = h*s -0.5s
  (s = scale = 0.01)



- Assume height field is locally constant
  - Lookup height field at To
- Trace ray from To to eye with eye vector V to height and add offset:
  - Tn = To + (hn * Vx,y/Vz)

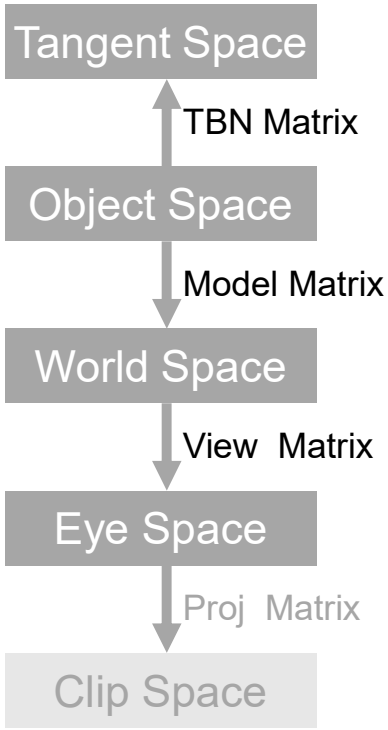
Offset Limited Parallax-Normal Mapping

- Problem: At steep viewing angles, Vz goes to zero
  - Offset values approach infinity
  - Solution: we leave out Vz division:
    Tn = To + (hn * Vx,y)
- Effect: offset is limited



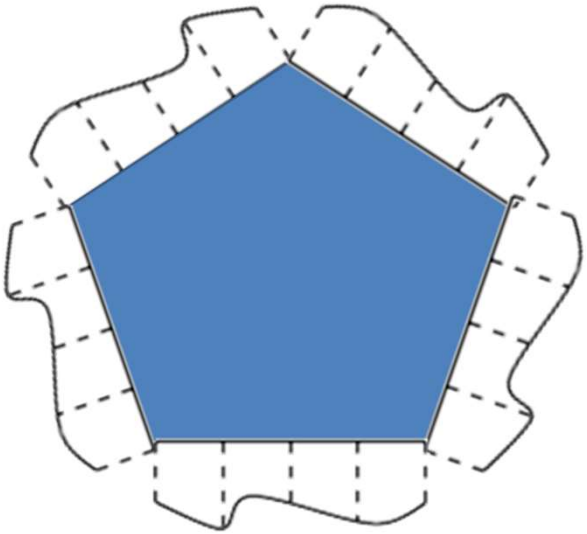
Coordinate Systems

- Problem: where to calculate lighting?
- Object coordinates
  - Native space for normals (N)
- World coordinates
  - Native space for light vector (L), env-maps
  - Not explicit in OpenGL!
- Eye Coordinates
  - Native space for view vector (V)
- Tangent Space
  - Native space for normal maps



Why Tangent Space?

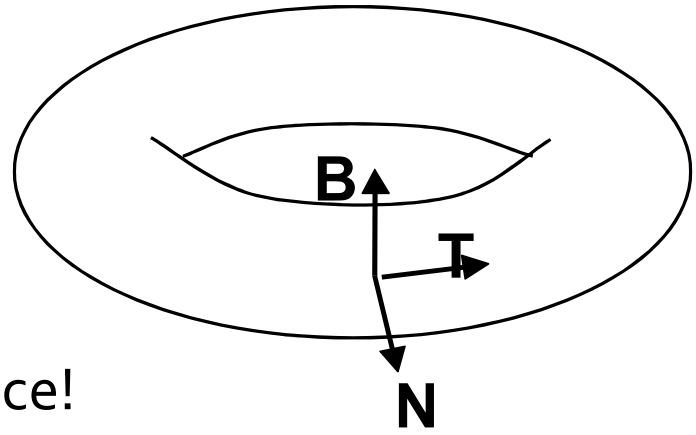
- Normals are in coordinate system of texture
- Not in coordinate system of model



Tangent Space

- Concept from differential geometry
- Set of all tangents on a surface
- Orthonormal coordinate system (frame) for each point on the surface:

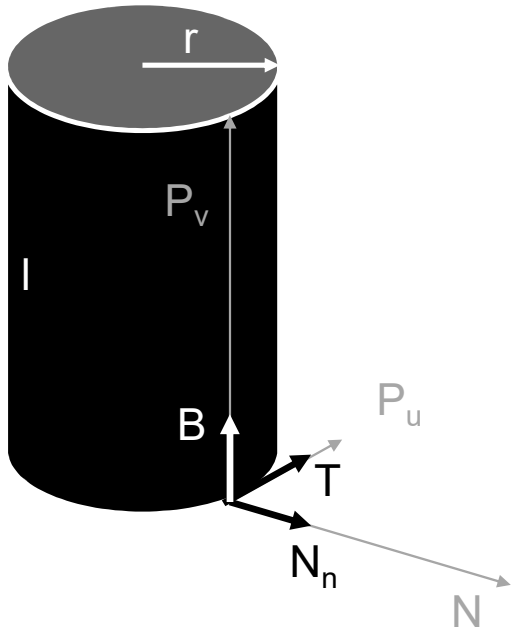
  $N_n(u,v) = P_u \times P_v / |P_u \times P_v|$
  $T = P_u / |P_u|$
  $B = N_n \times T$



- A natural space for normal maps
  - Vertex normal N = (0,0,1) in this space!

Parametric Example

- Cylinder Tangent Space:
- $N_n(u,v) = P_u \times P_v / |P_u \times P_v|$
  $T = P_u / |P_u|$
  $B = N_n \times T$
- Tangent space matrix:
  TBN column vectors
  - Transforms from tangent into object space



Creating Tangent Space

- Trivial for analytically defined surfaces
  - Calculate P_u, P_v at vertices
- Use ***texture space*** for polygonal meshes
  - P_u aligned with u direction and P_v with v
  - Origin is texture coordinate of vertex
- Transformation from object space to tangent space (if TBN is a orthonormal matrix)

$$\begin{matrix} L_{tx} & L_{ty} & L_{tz} \end{matrix} = \begin{matrix} L_{ox} & L_{oy} & L_{oz} \end{matrix} \begin{matrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{matrix}$$

Creating Tangent Space for a Mesh

- Calc tangent for a triangle P_0, P_1, P_2
  - With texture coordinates (u_0, v_0), (u_1, v_1), (u_2, v_2)
  - Work relative to P_0
    - $Q_1 = P_1 - P_0$ $\quad (s_1, t_1) = (u_1 - u_0, v_1 - v_0)$
    - $Q_2 = P_2 - P_0$ $\quad (s_2, t_2) = (u_2 - u_0, v_2 - v_0)$
  - Need to solve
    - $Q_1 = s_1T + t_1B$
    - $Q_2 = s_2T + t_2B$
    - Solve linear System with 6 unknowns and 6 equations
- Vertex tangents by averaging tangents of all triangles sharing the vertex

Creating Tangent Space for a Mesh

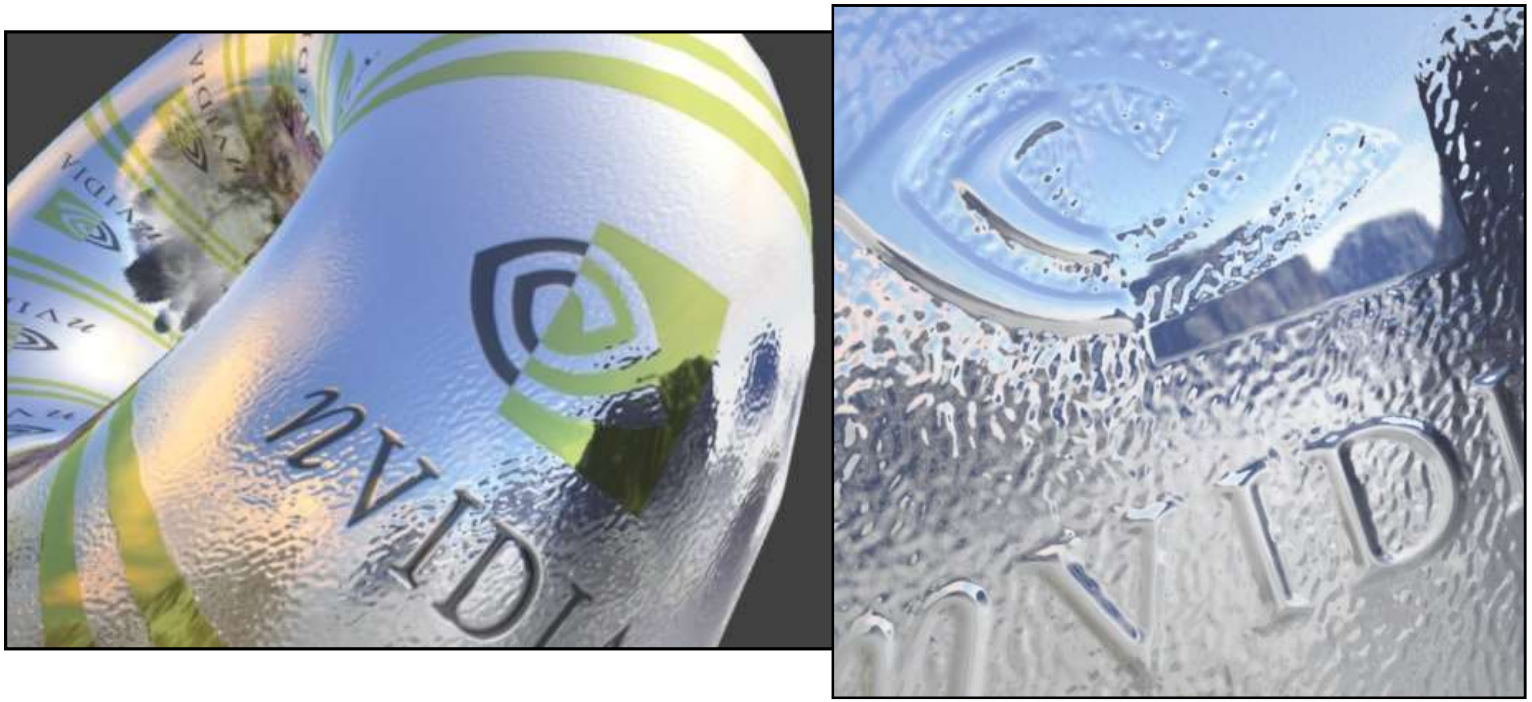
- Need to orthogonalize resulting tangents for inverting with transpose
  - Gram-Schmidt
- If only one tangent is stored we need to account for handedness.
- Code and details at http://www.terathon.com/code/tangent.html

Fast Algorithm (Tangent Space)

- For each vertex
  - Transform light direction L and eye vector V to tangent space and normalize
  - Compute normalized half vector H
- For each fragment
  - Interpolate L and H
  - Renormalize L and H
  - Fetch N′ = texture(s, t) (normal map)
  - Use N′ in shading

Normal Mapping + Environment Mapping

- Normal and parallax mapping combines beautifully with environment mapping



EMNM (World Space)

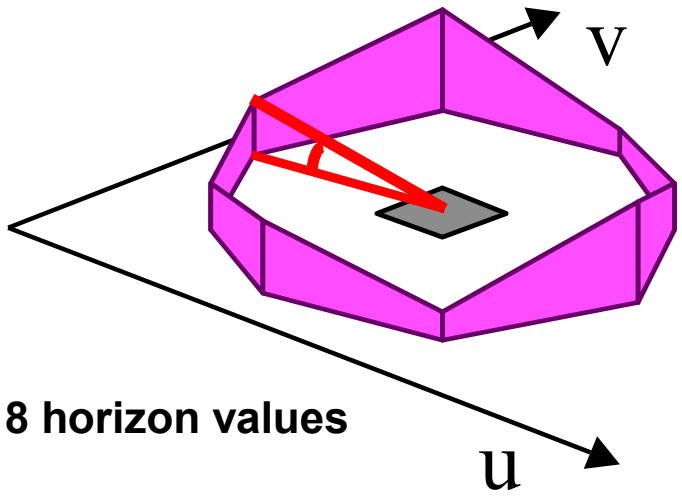
- For each vertex
  - Transform V to world space
  - Compute tangent space to world space transform (T, B, N)
- For each fragment
  - Interpolate and renormalize V
  - Interpolate frame (T, B, N)
  - Lookup N′ = texture(s, t)
  - Transform N′ from tangent space to world space
  - Compute reflection vector R (in world space) using N′
  - Lookup C = cubemap(R)

Normal and Parallax Normal Map Issues

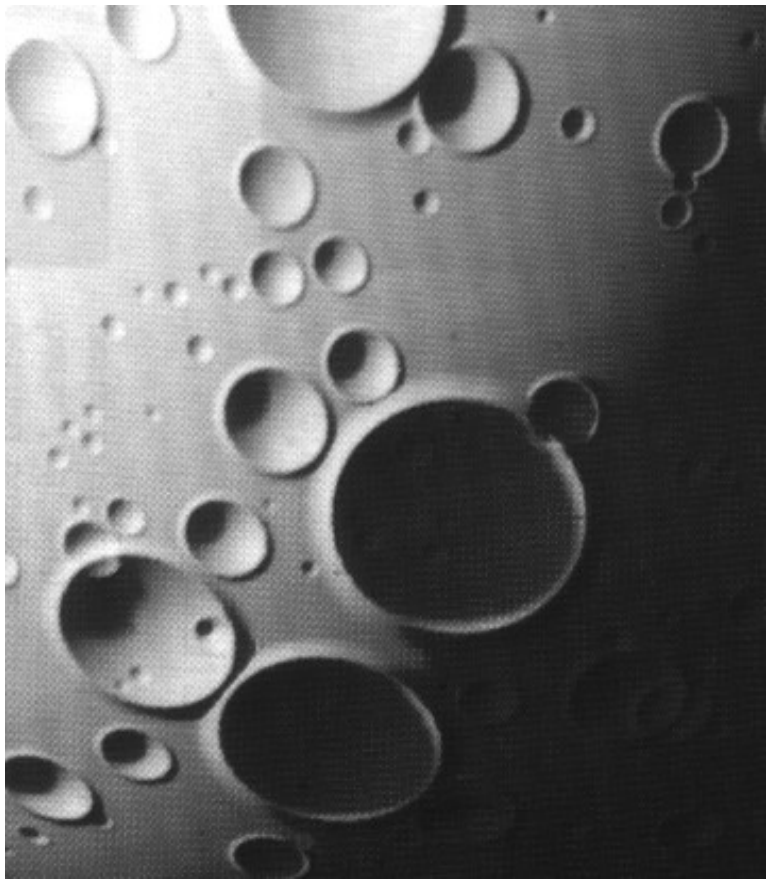
- Artifacts
  - No inter-shadowing
  - Silhouettes still edgy
  - No parallax for normal mapping
- Parallax normal mapping
  - No occlusion, just distortion
  - Not accurate for high frequency height fields
    (local constant height field assumption does not work)
  - No silhouettes

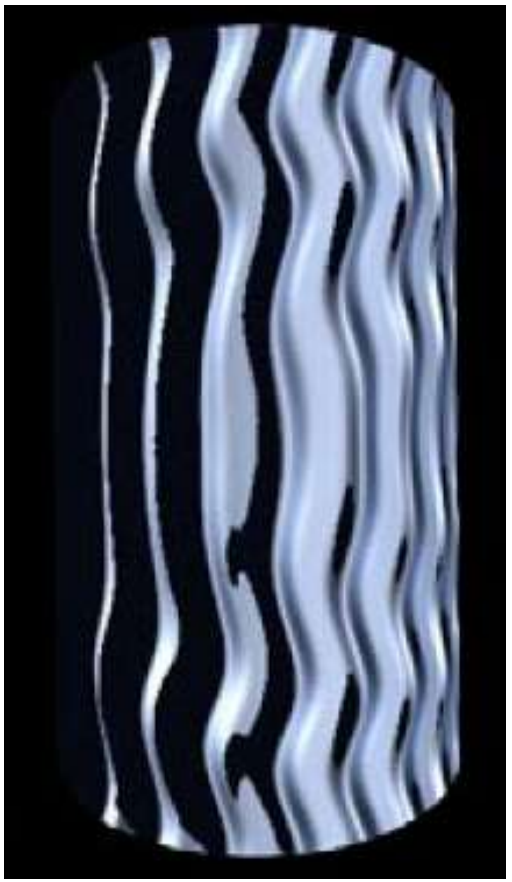
Horizon Mapping

- Improve normal mapping with (local) shadows
- Preprocess: compute n horizon values per texel
- Runtime:
  - Interpolate horizon values
  - Shadow accordingly



Horizon Mapping Examples





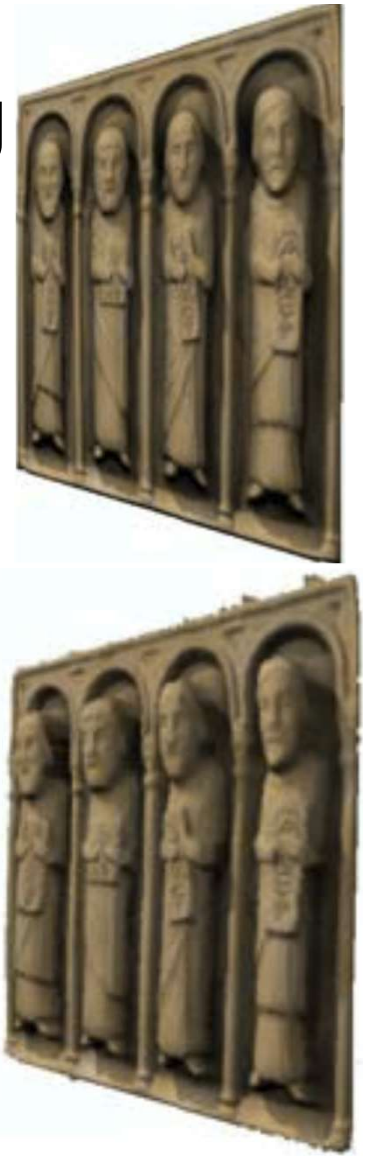
Relief Mapping

Relief Texture Mapping

- Uses image warping techniques and per-texel depth to create the illusion of geometric detail

Relief Texture Mapping

- Rendering of a height field requires search for closest polygon along viewing ray
- Two-pass method:
  - Convert height field to 2D texture
    using forward projection
  - Render texture
- Texels move horizontal and vertical in texture space based on their orthogonal displacements and the viewing direction



Relief Mapping Examples

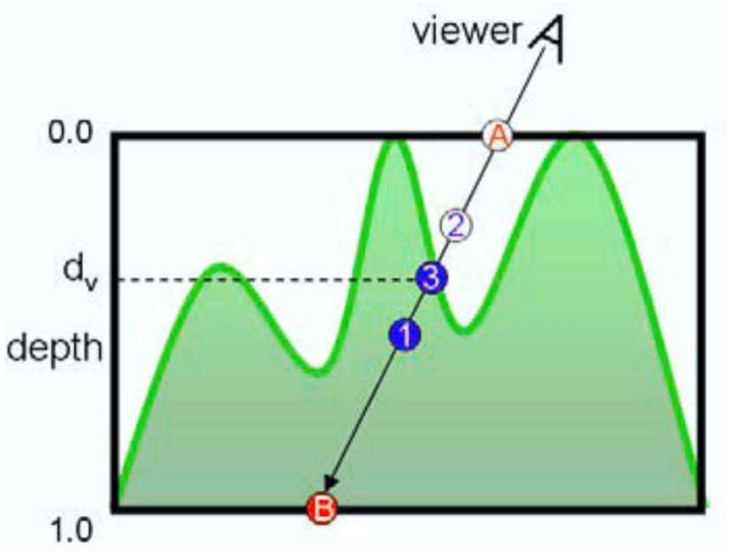
Texture mapping

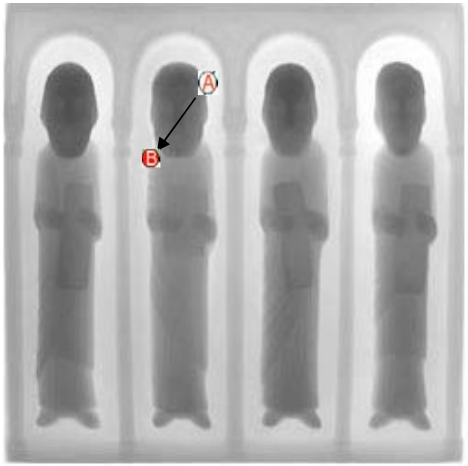
Parallax mapping

Relief mapping

Mapping Relief Data

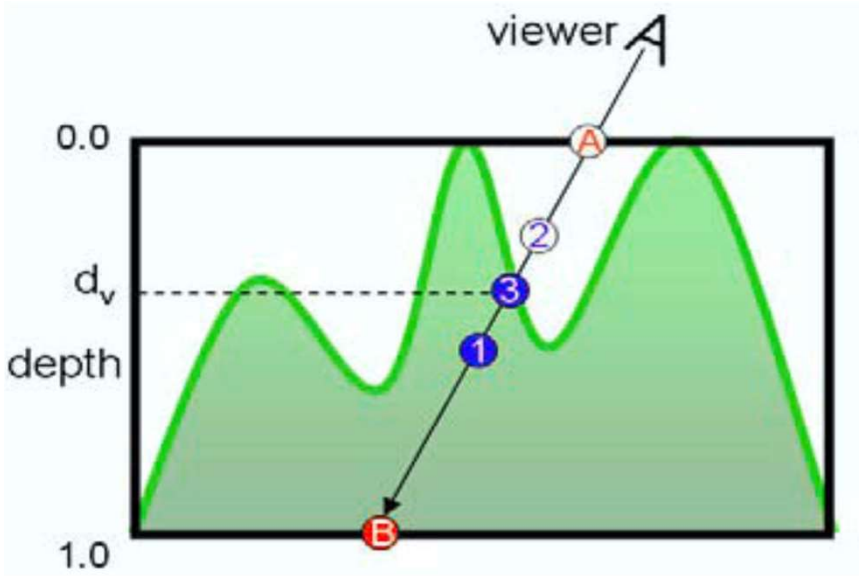
- Compute viewing direction, VD
- Transform VD to tangent space of fragment
- Use VD' and texture coords (s,t) to compute the texture coords where VD' hits depth of 1





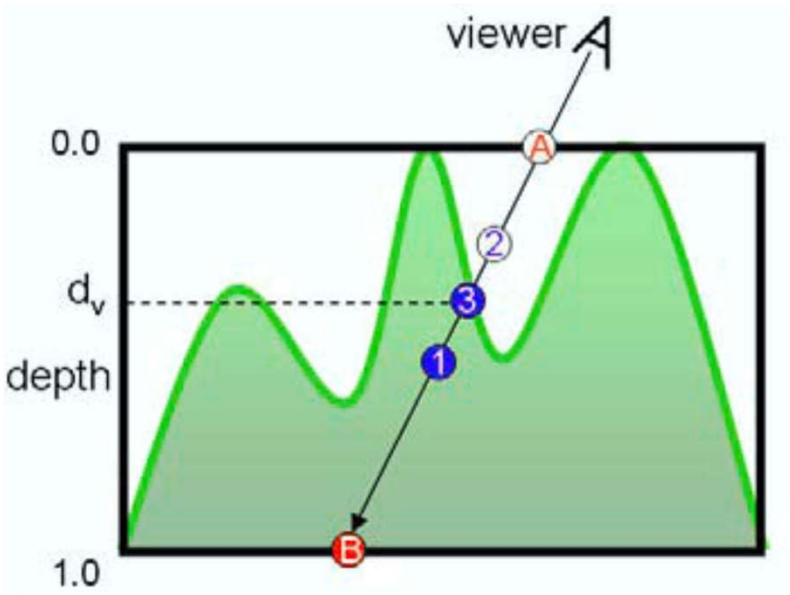
Mapping Relief Data

- Compute intersection between VD' and height-field surface using binary search starting with A and B
- Perform shading of the fragment using the attributes associated with the texture coordinates of the computed intersection point.



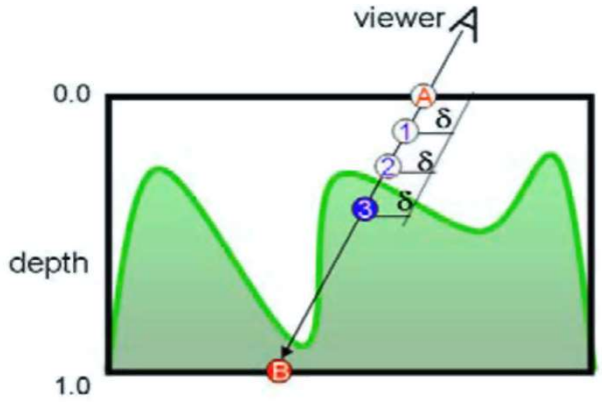
Binary Search

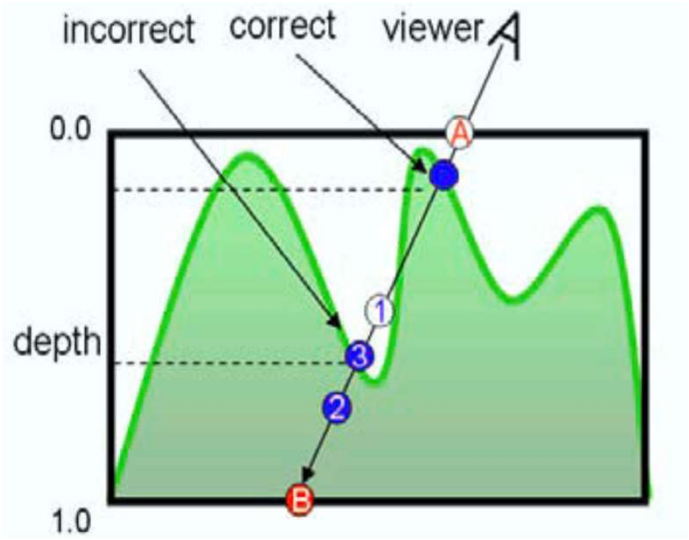
- Start with A-B line
- At each step:
  - Compute middle of interval
  - Assign averaged endpoint texture coordinates and depth
  - Use averaged tex coords to access depth map
  - If stored depth value is less than computed depth value, the point is inside the surface
  - Proceed with one endpoint in and one out



Combined Search

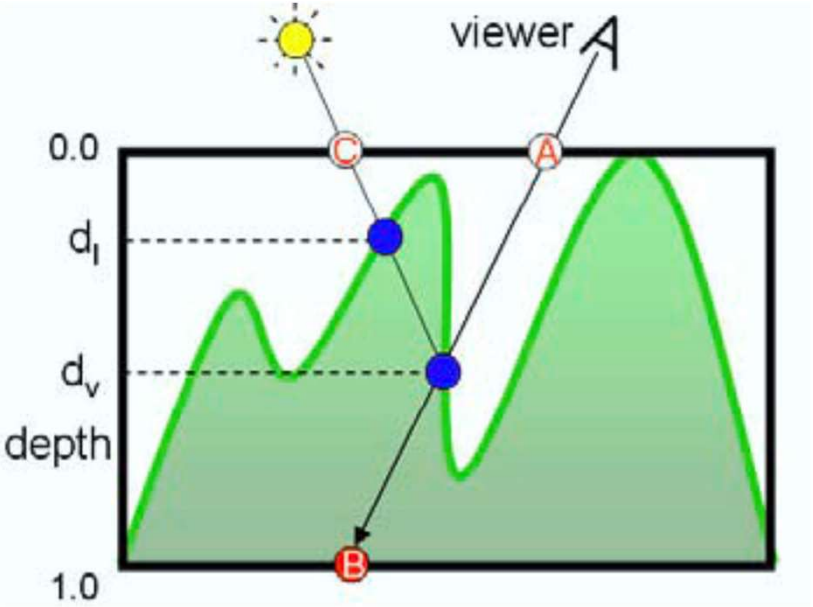
- To find first point under surface, start at A, advance ray by δ
- δ is a function of the angle between VD′ and interpolated fragment normal
- Proceed with binary search
  (with less iterations)





Shadowing

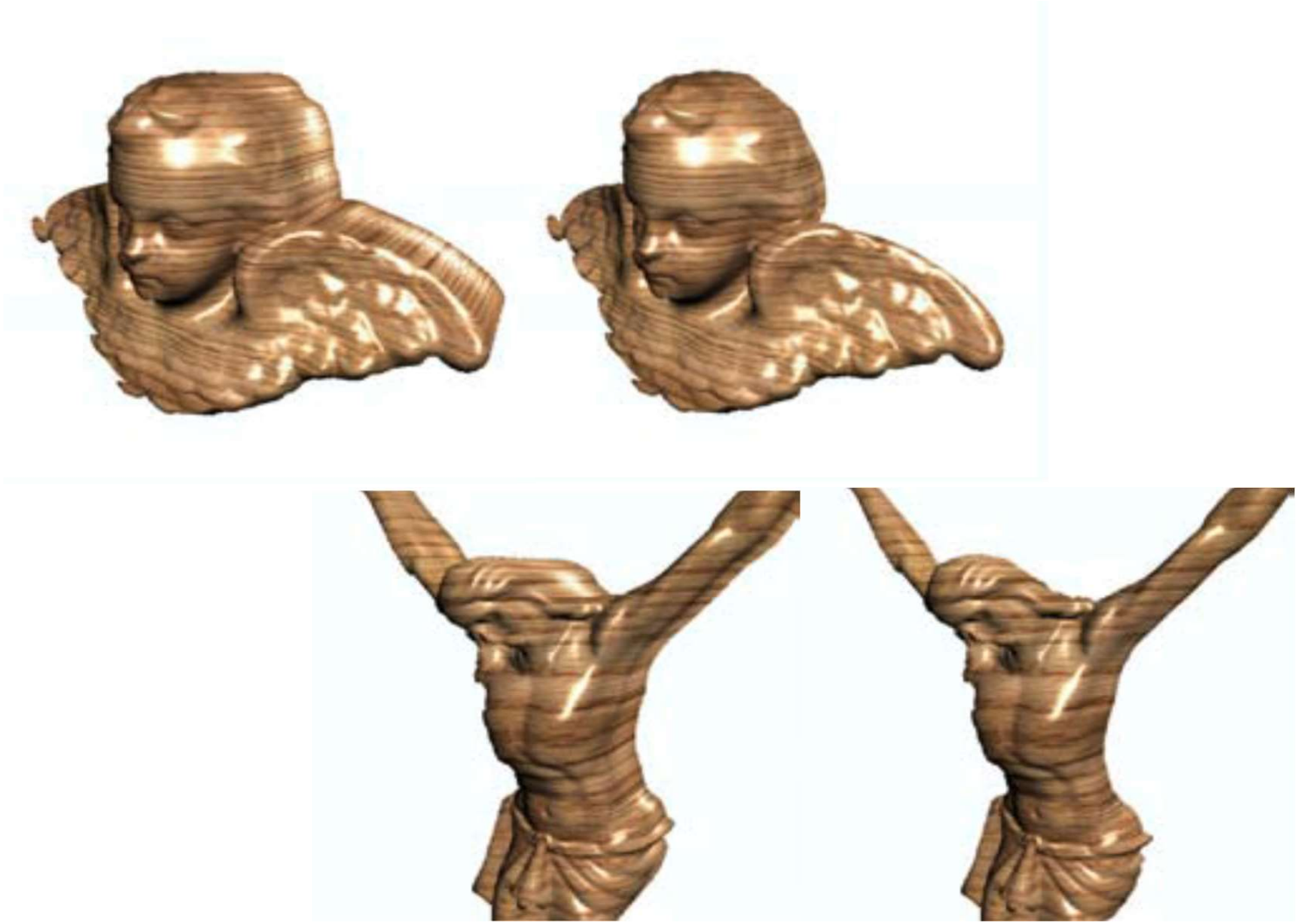
- Visibility problem
- Determine if light ray intersects surface
- Do not need to know the exact point



Dual Depth Relief Textures

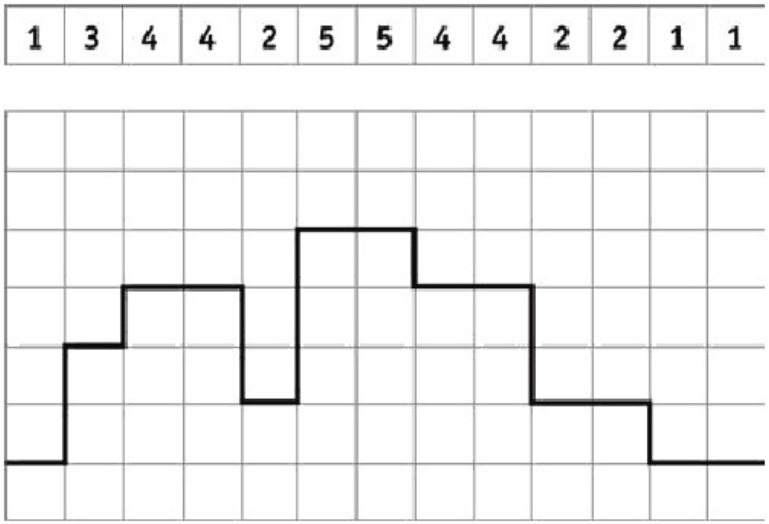
- Represent opaque, closed surfaces with only one texture
- Second "back" layer is not used for rendering, but as a constraint for ray-height-field intersection

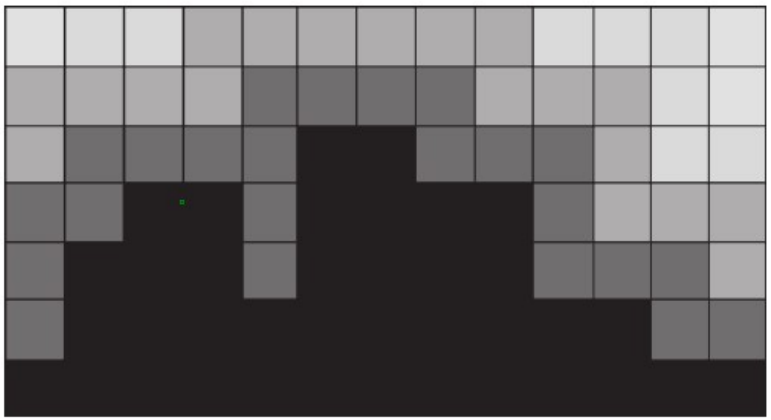
Dual Depth Relief Textures



Sphere Tracing

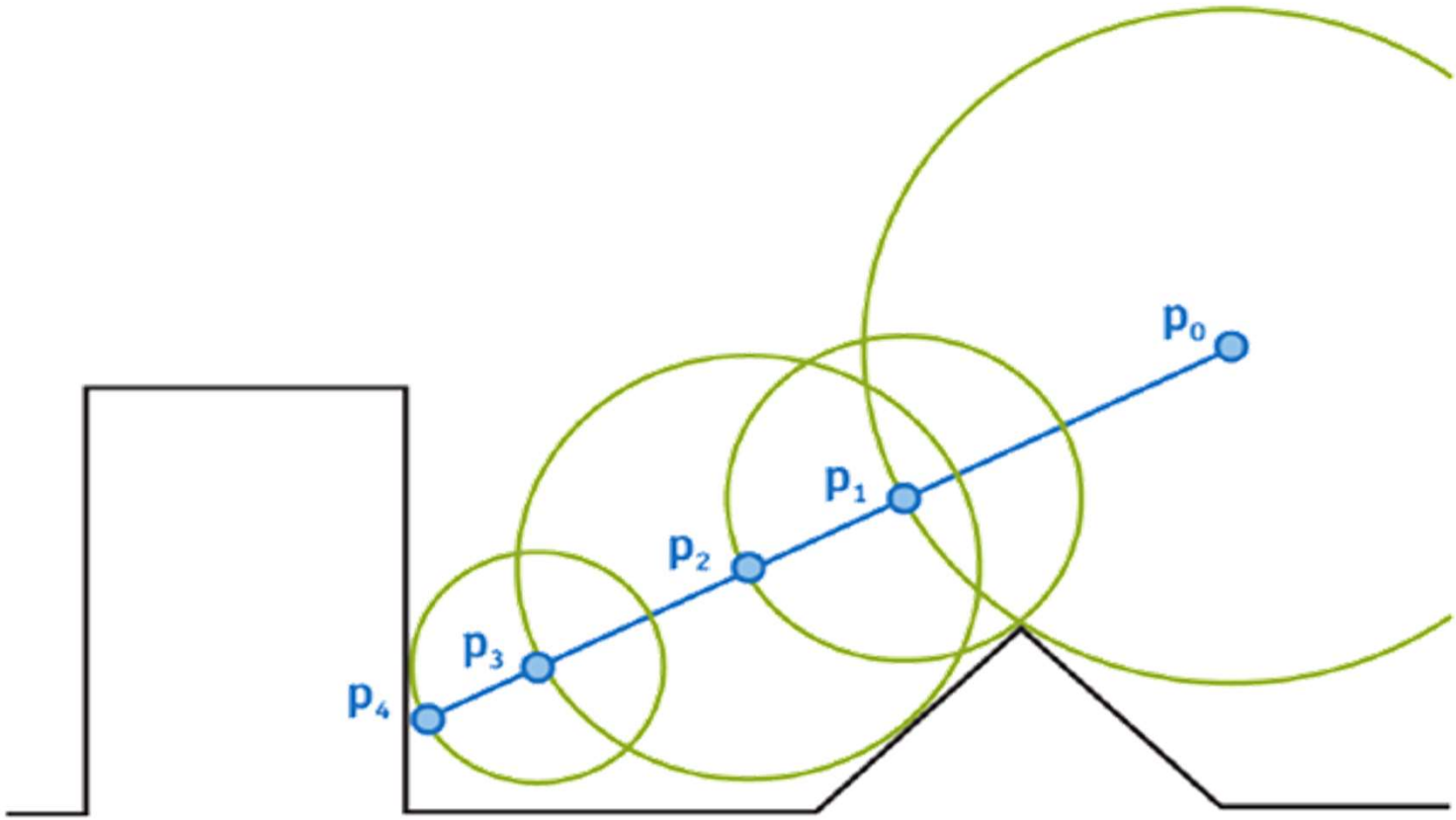
- Store distance to closest surface in 3D map





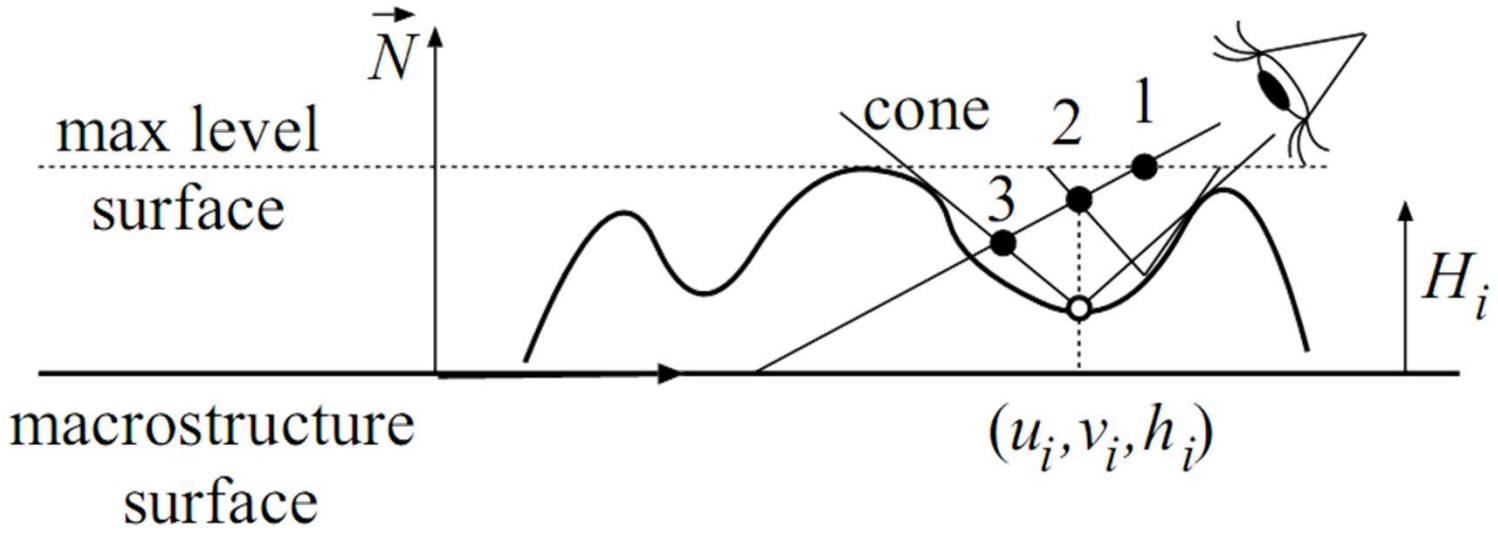
Sphere Tracing

- Distance is sphere radius for search step



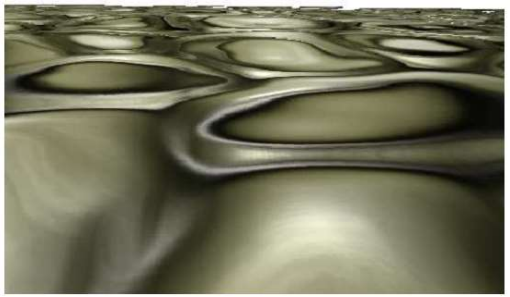
Cone Stepping

- Texel stores cone of empty space above
- Only store opening angle (2D texture suffices!)



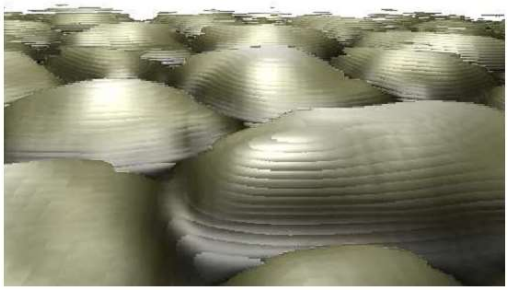
texture mapping

bump mapping

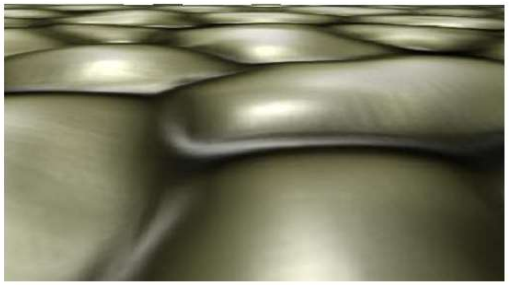


parallax mapping

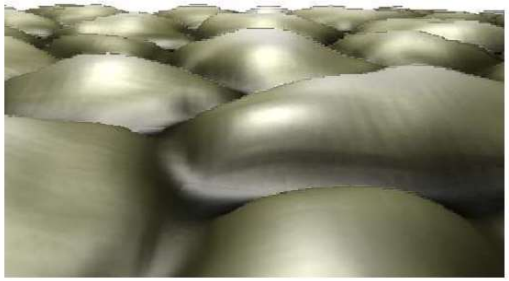
ray marching



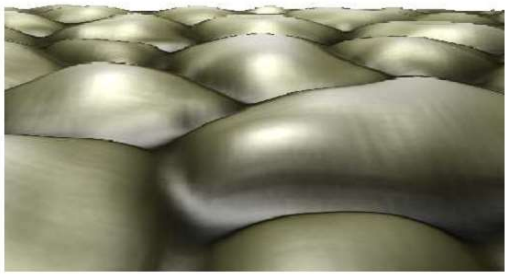
binary search



sphere tracing



relief mapping



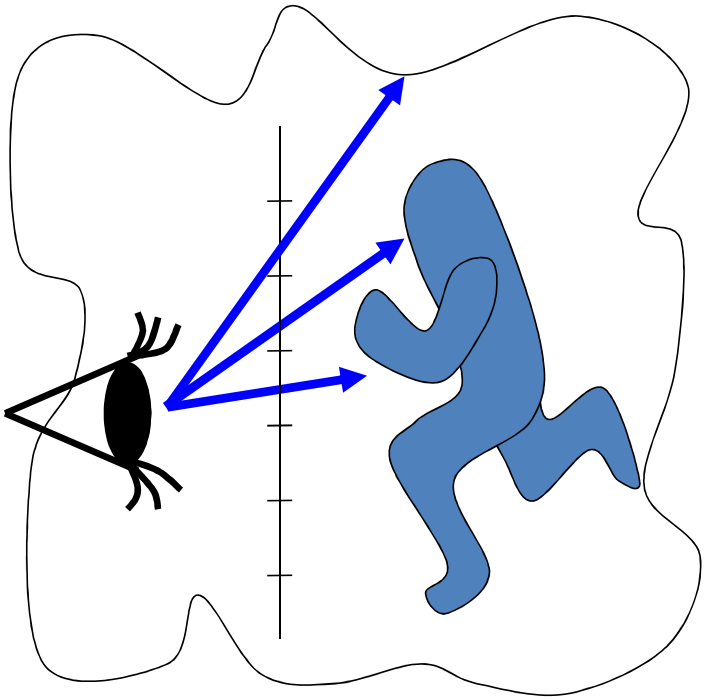
cone stepping

per-vertex displacement mapping

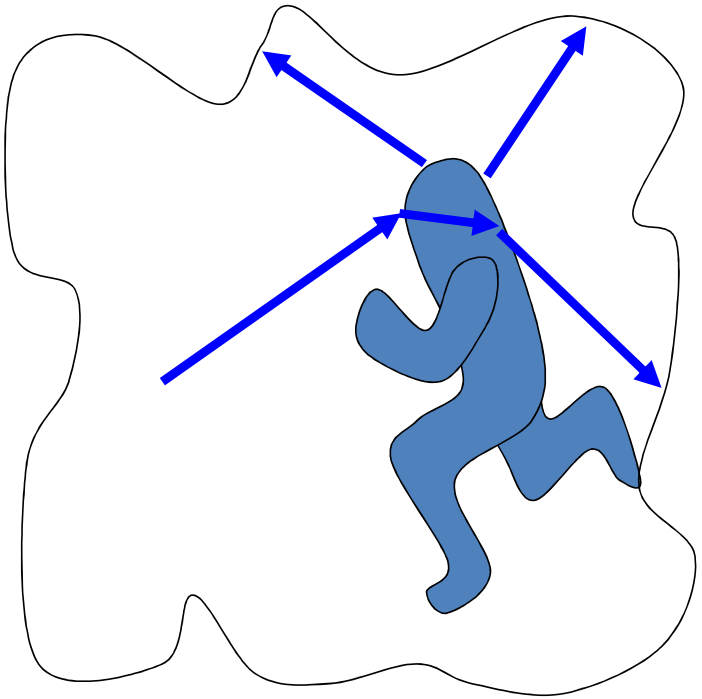
Speed considerations

- Parallax-normal mapping
  - ~ 20 ALU instructions
- Relief-mapping
  - Marching and binary search:
  - ~300 ALU instructions
  - + lots of texture lookups

Rasterization versus Ray-Tracing

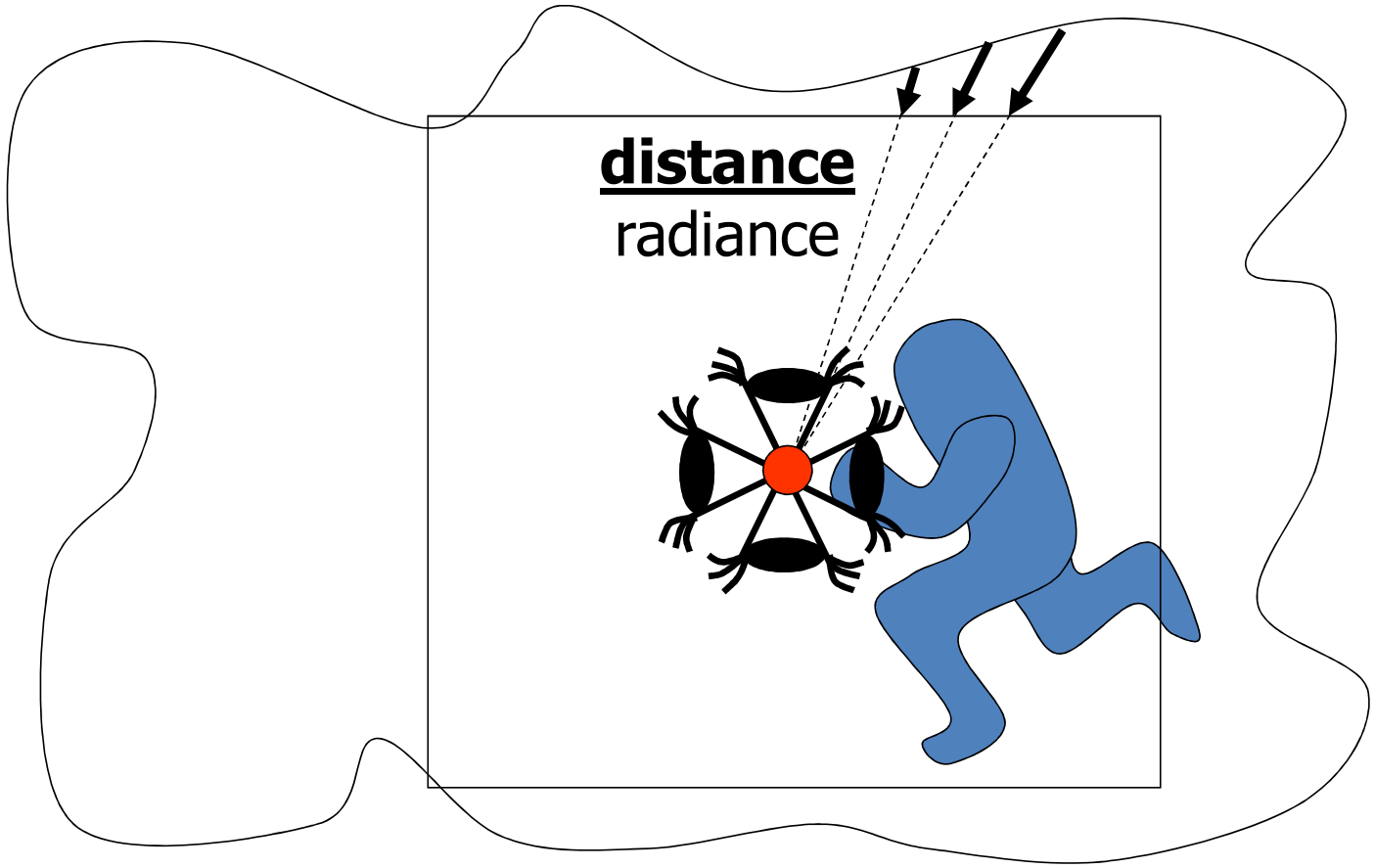


Inremental rendering
on the GPU

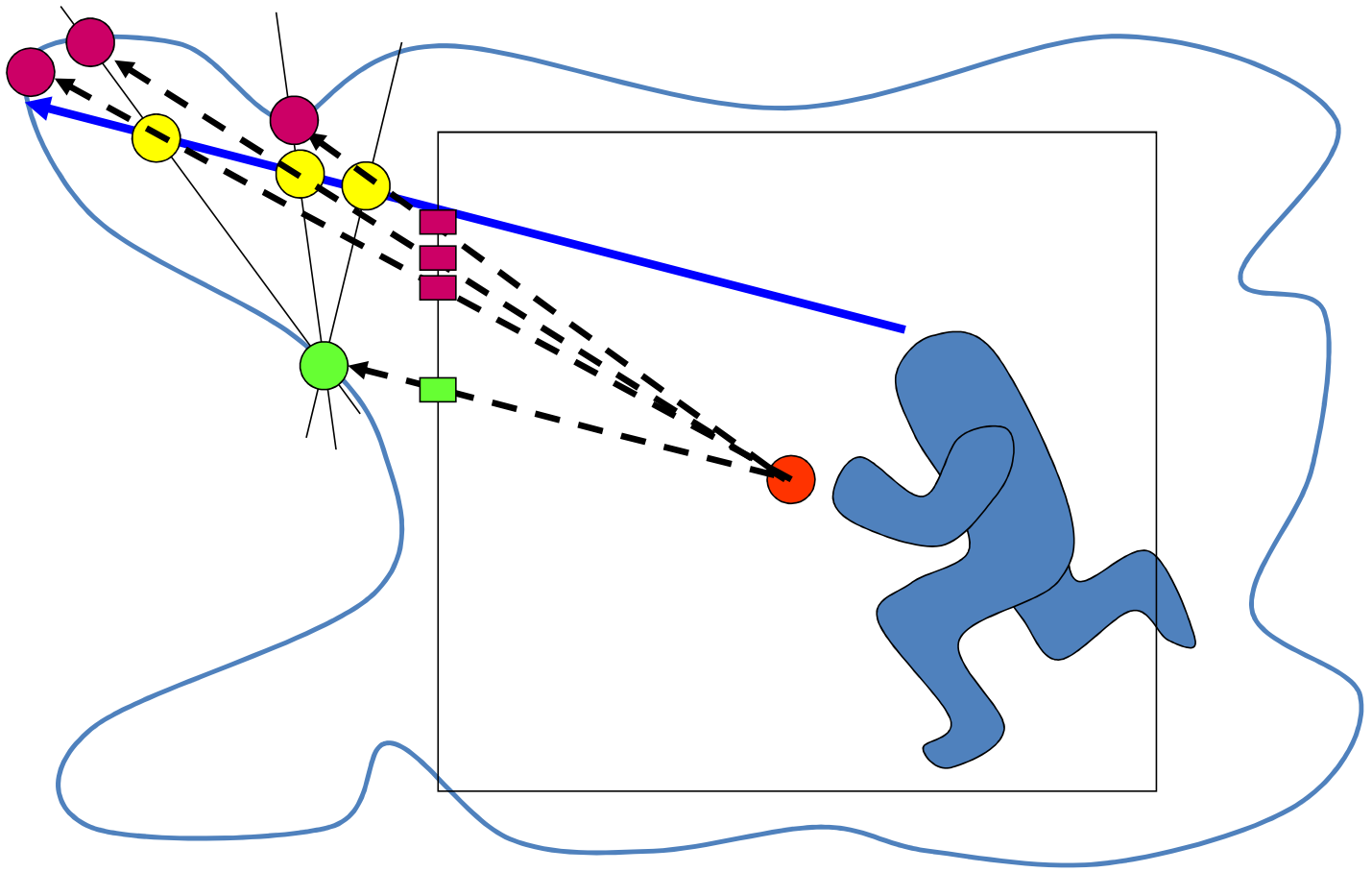


Non-coherent
Ray tracing

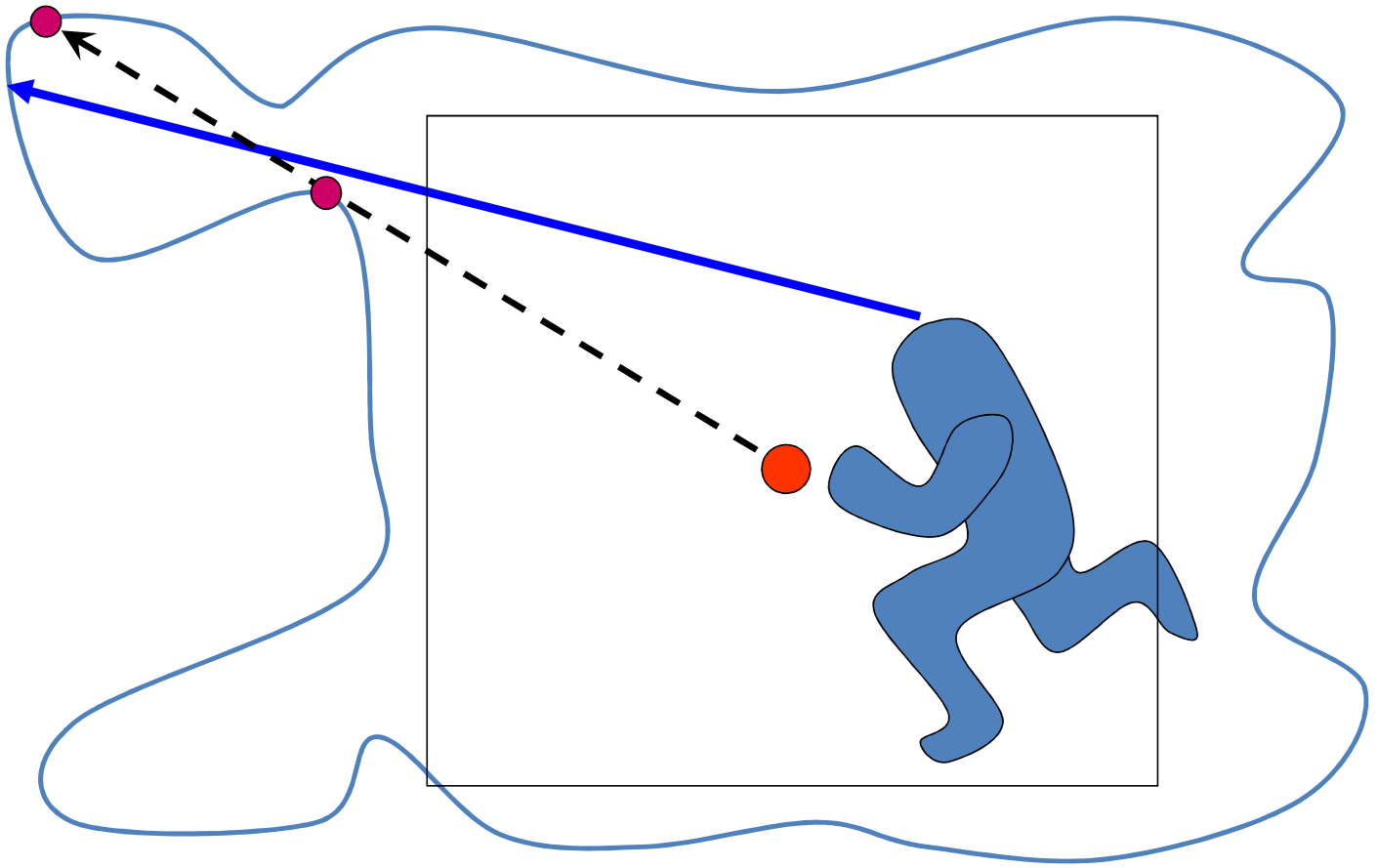
Distance Impostors



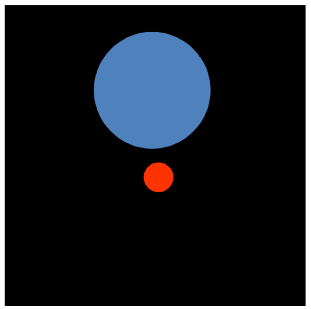
Ray-Tracing with Distance Impostors

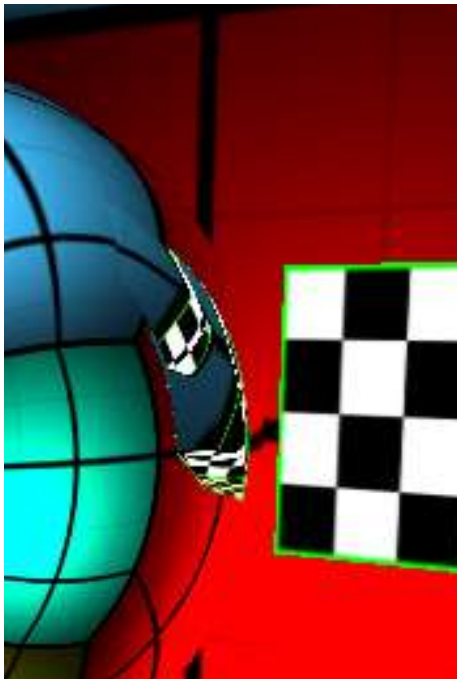


Approximation

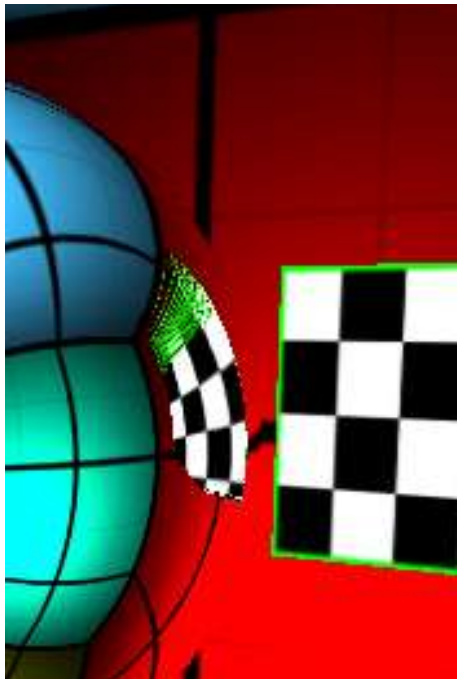


Approximation Error

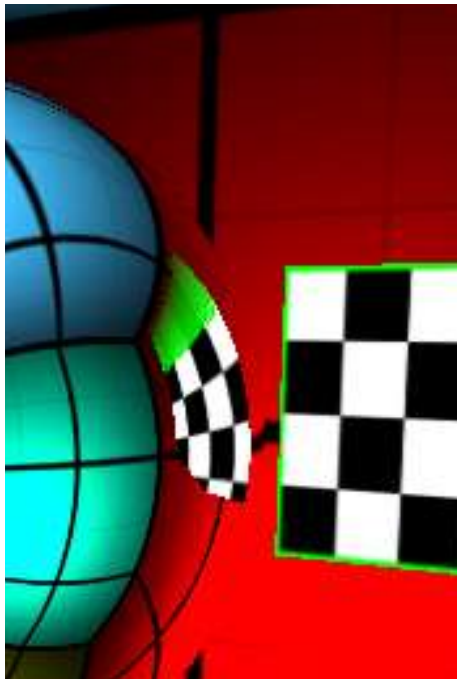




1 iteration

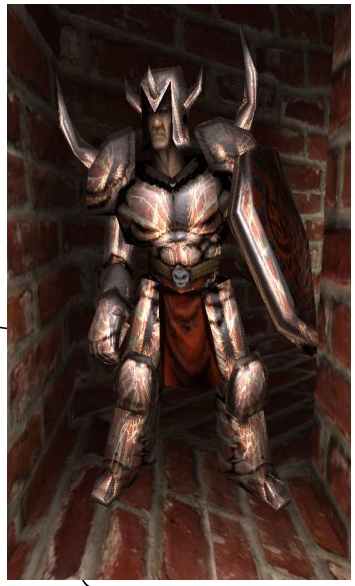


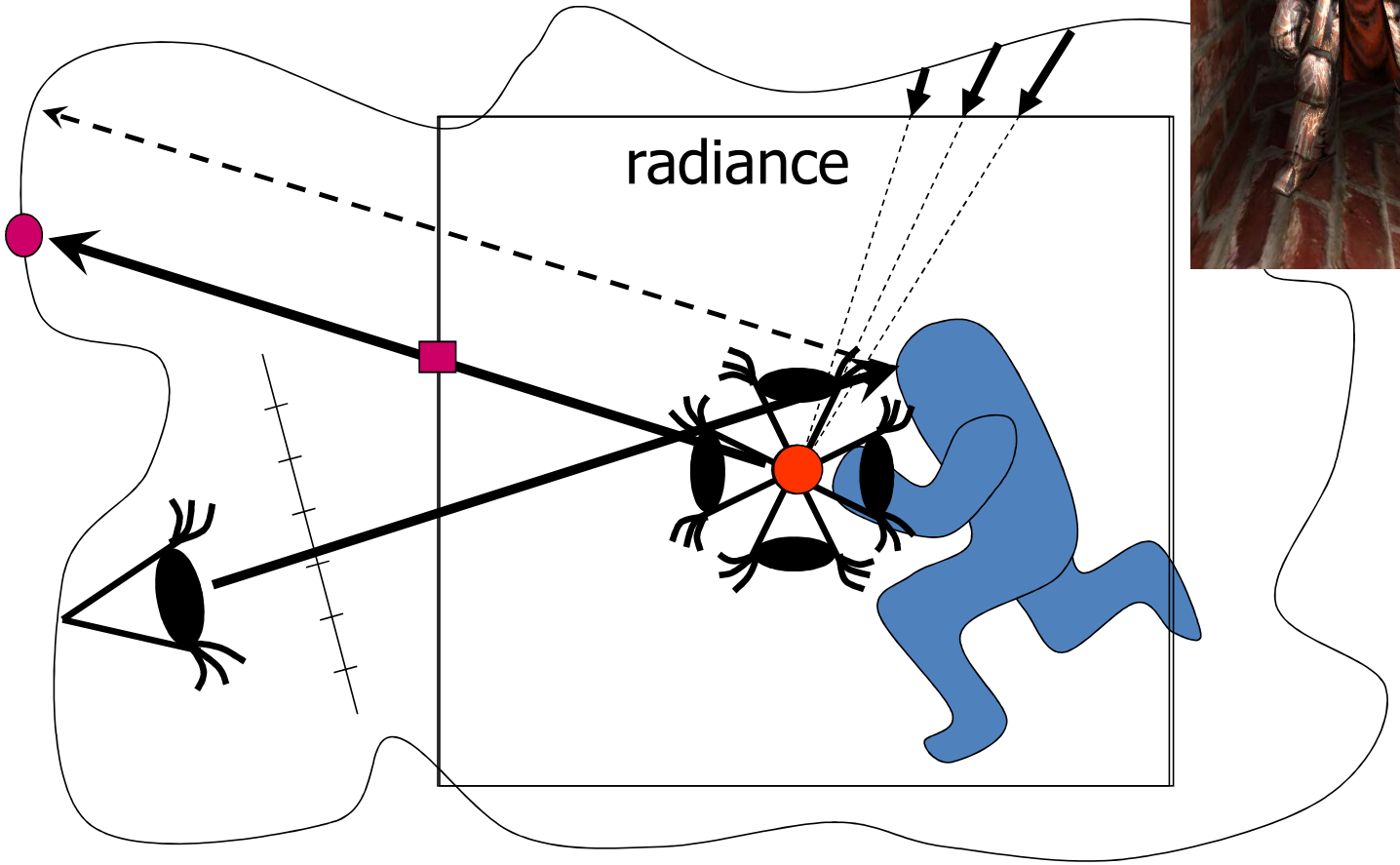
4 iterations



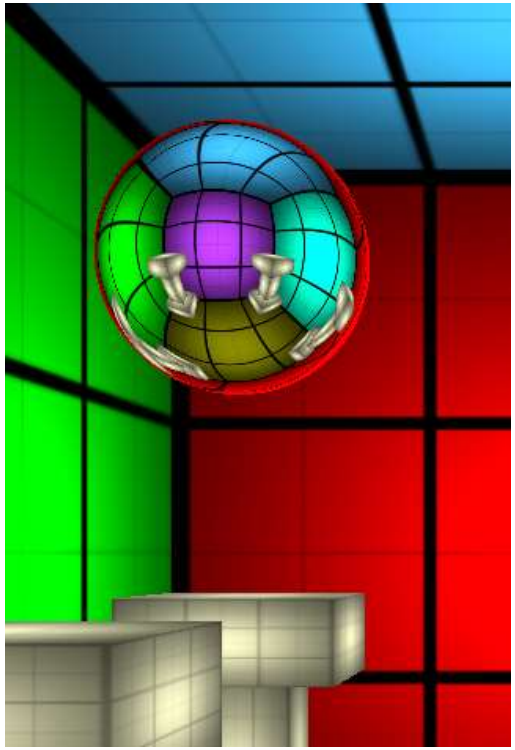
8 iterations

Reflections

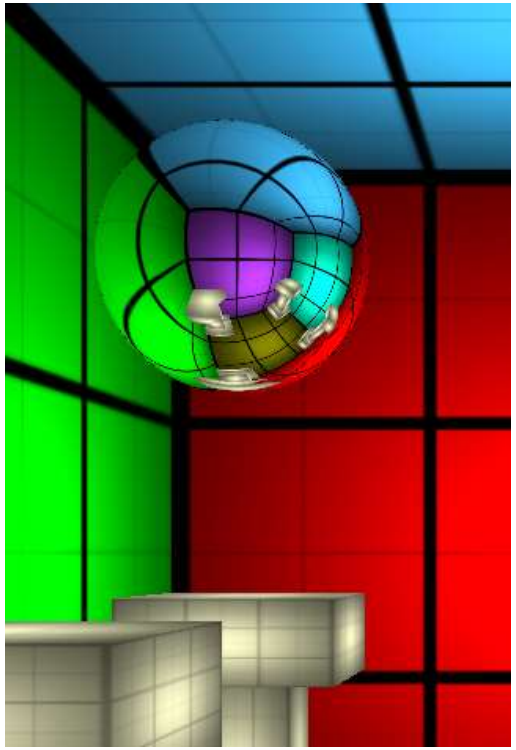




Problems of environment map based reflections

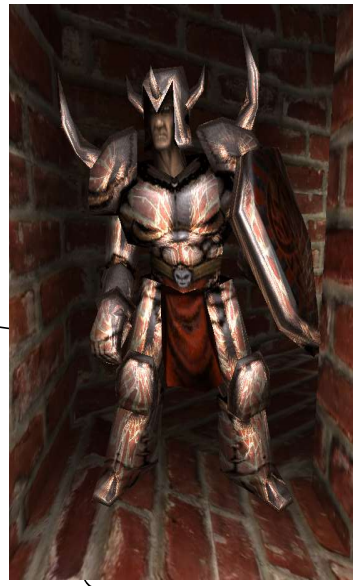


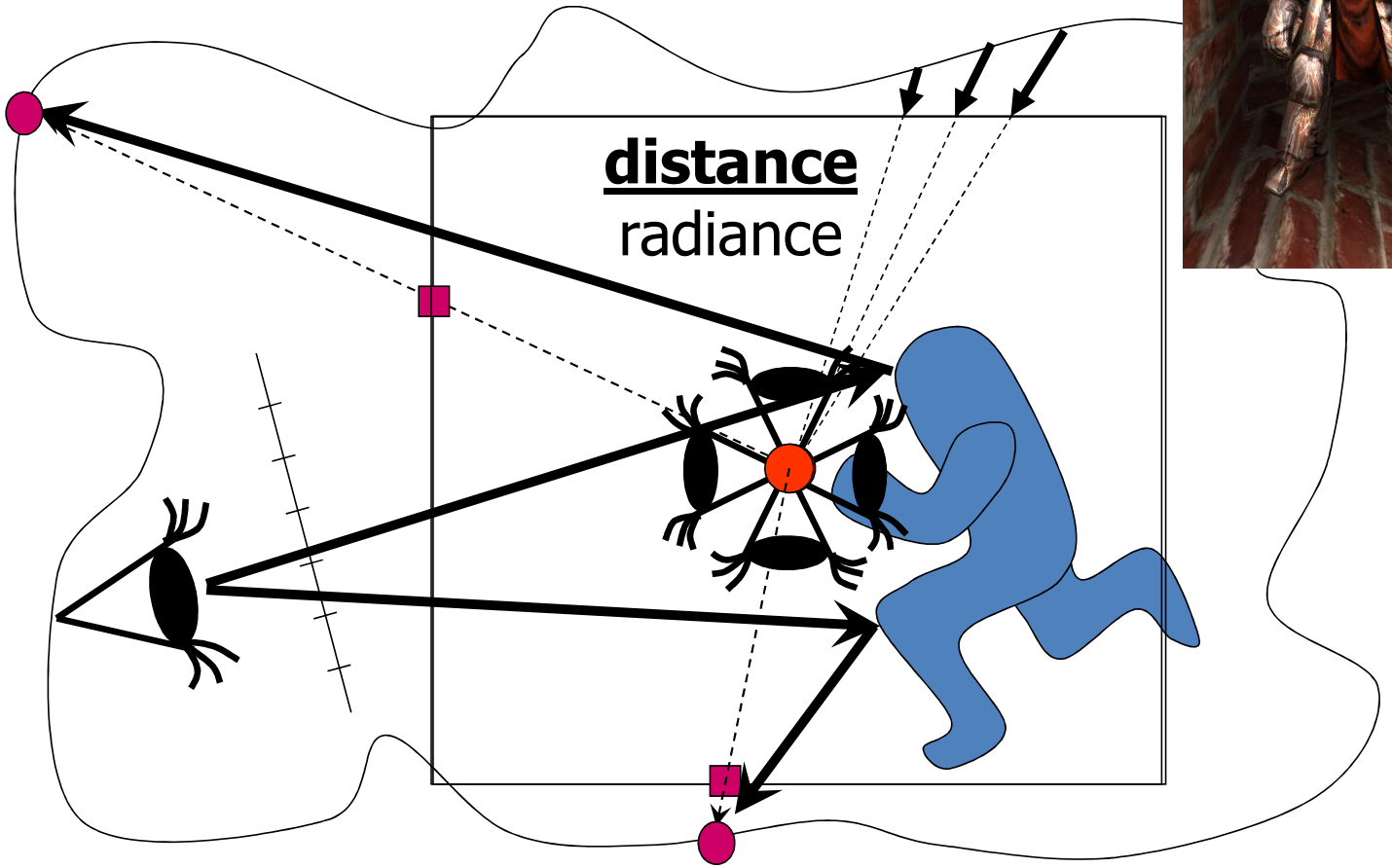
Environment map

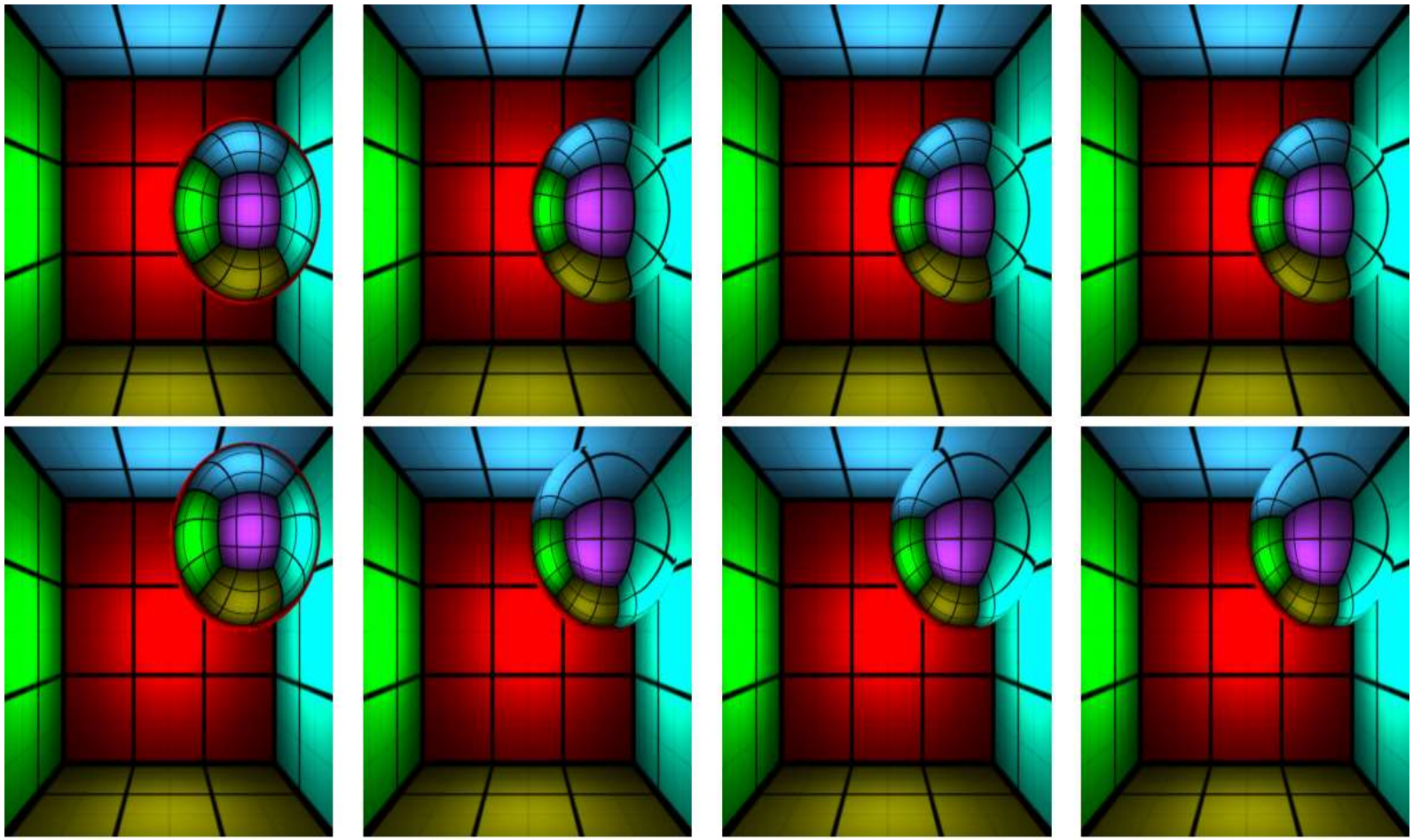


Reference

Localized Reflections







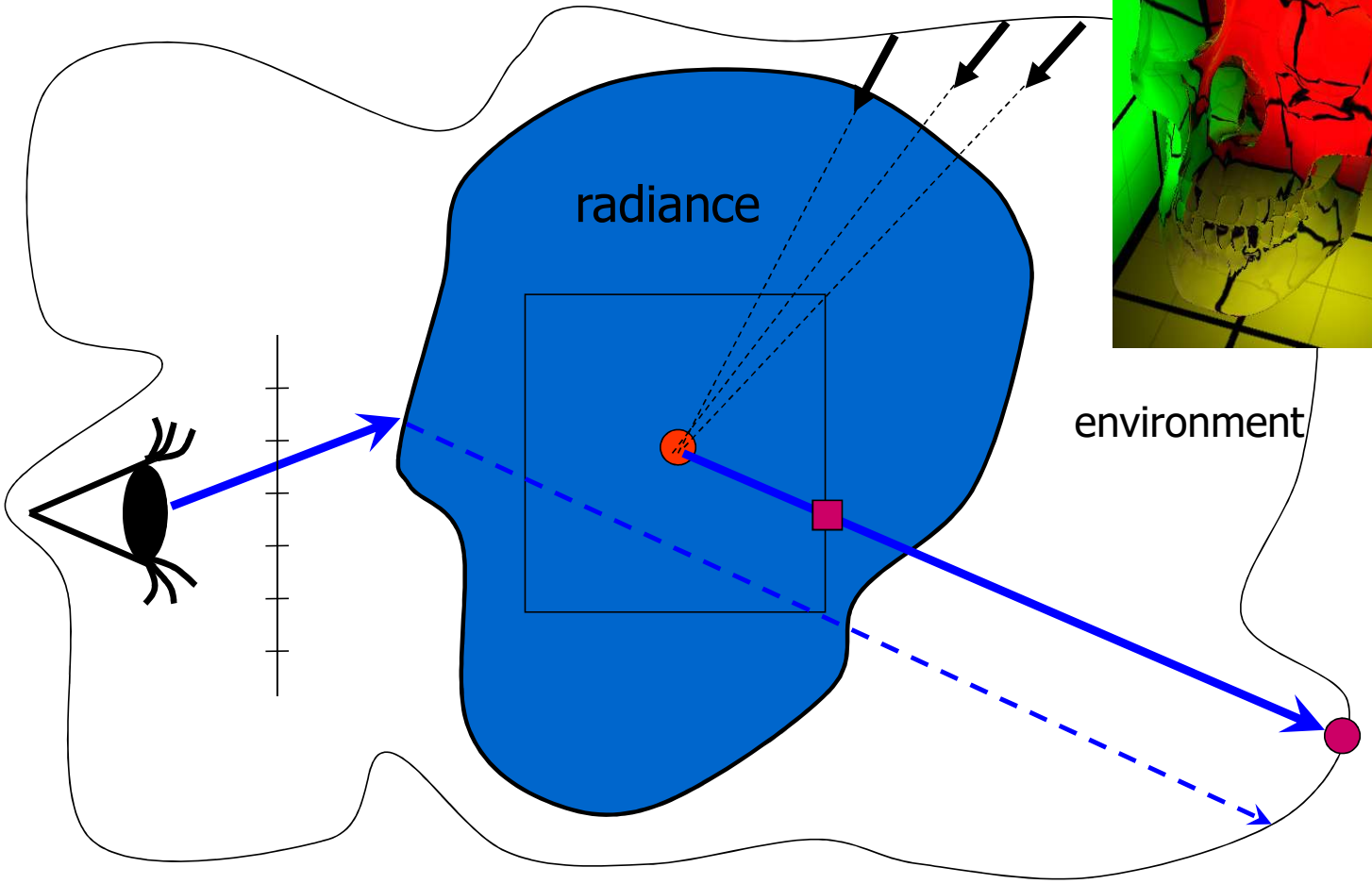
classical environment map
642 FPS

distance impostor
447 FPS

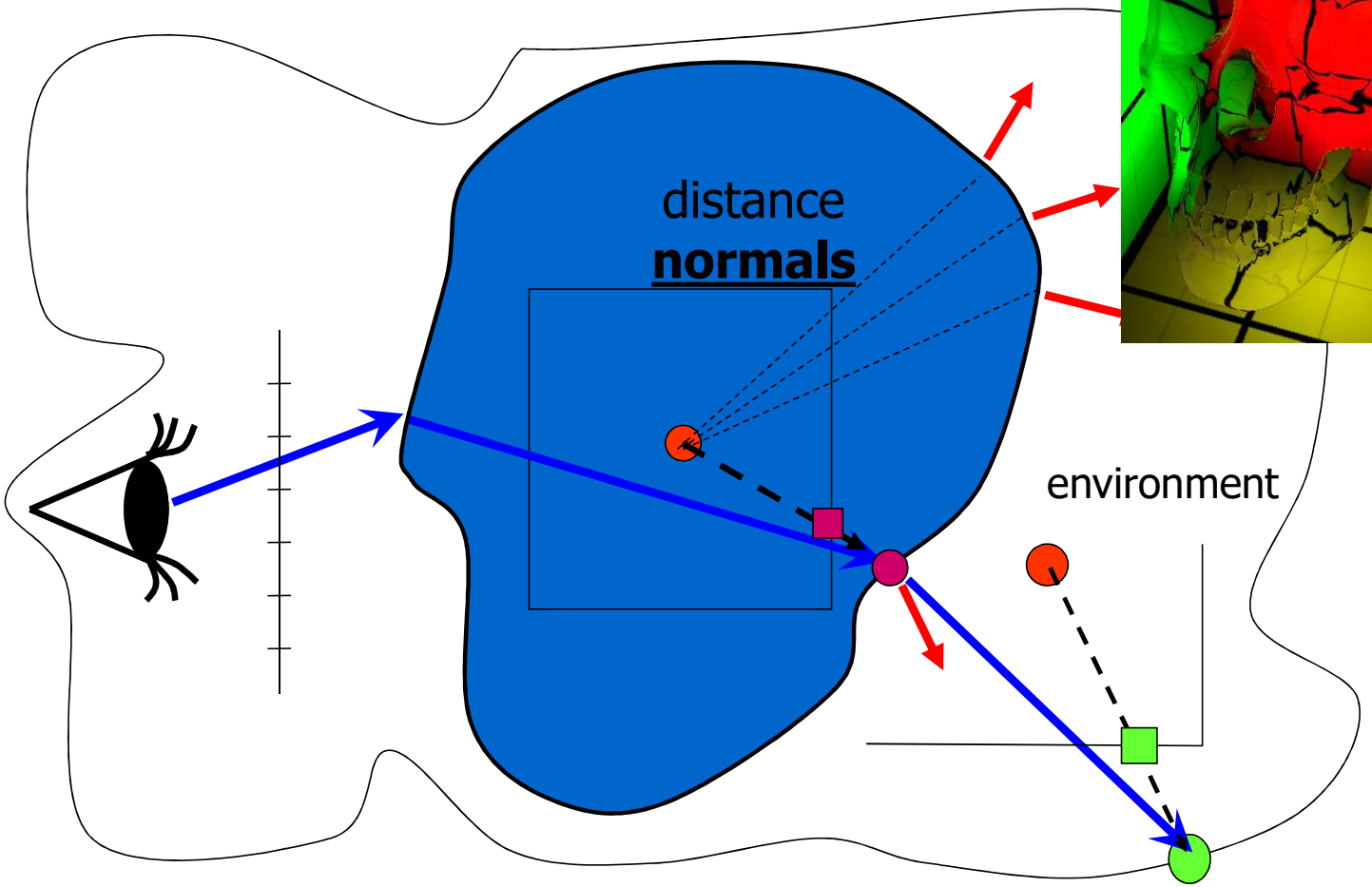
+1 iteration
323 FPS

ray traced reference

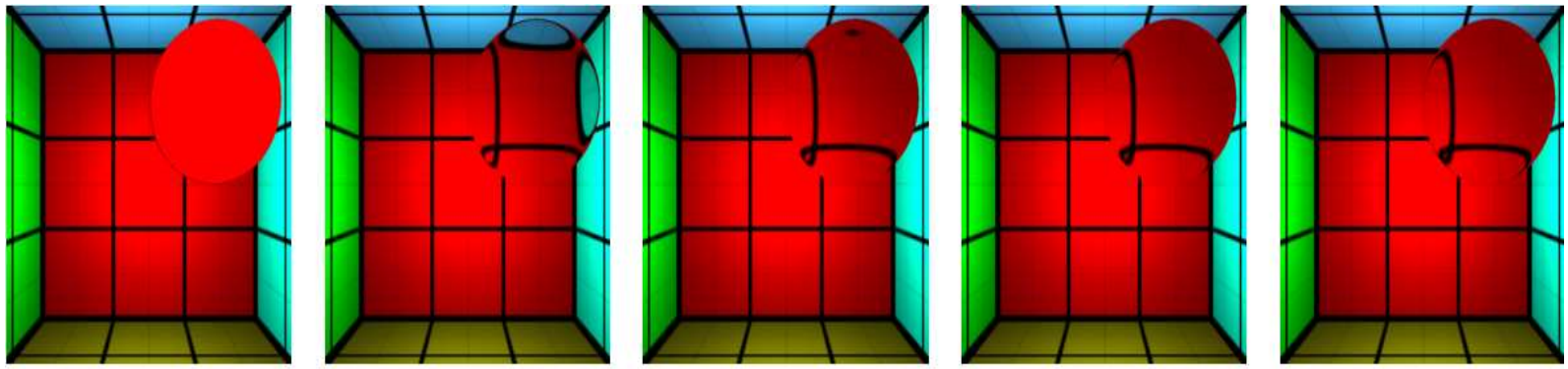
Refractions



Multiple Localized Refract



Multiple Localized Refractions



classical
environment map
804 FPS

distance impostor
single refraction
695 FPS

distance impostor
double refraction
508 FPS

+1 iteration
double refraction
249 FPS

ray traced
reference