Advanced Texturing

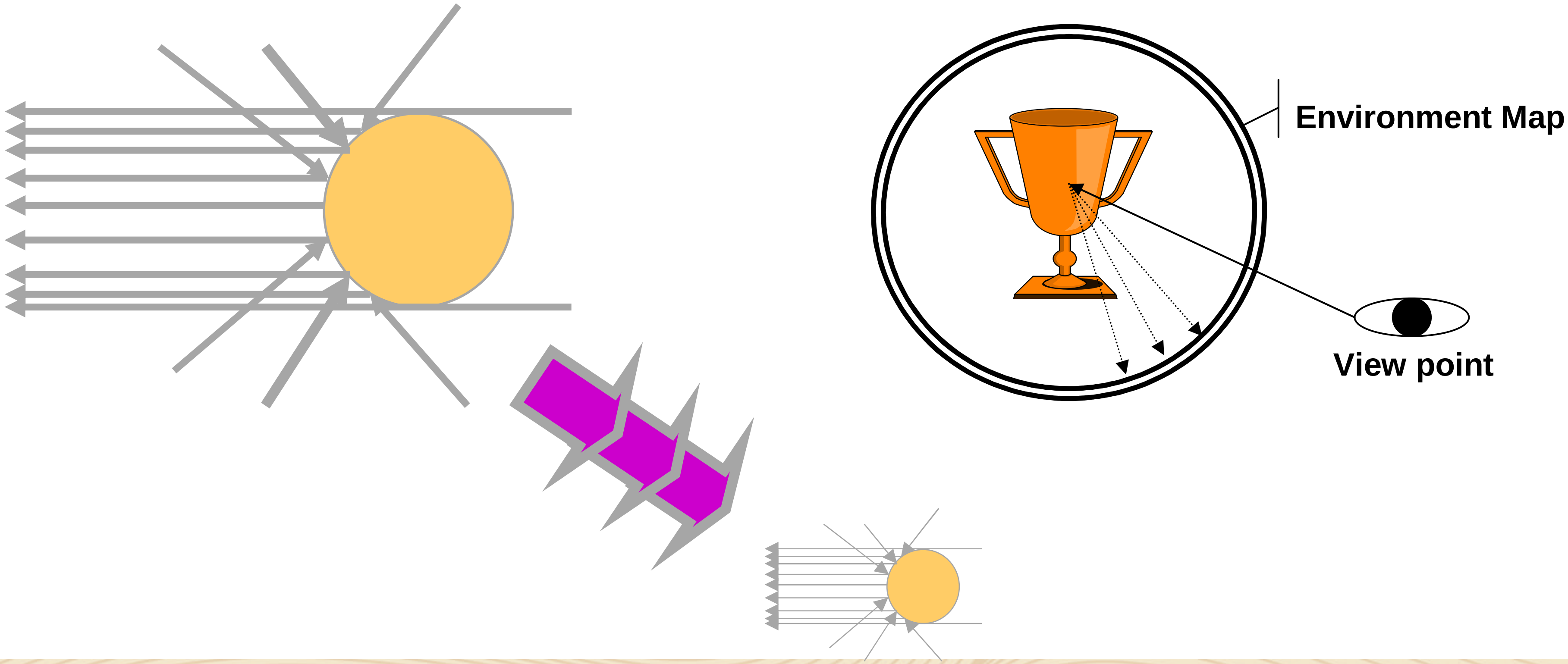
Environment Mapping

Environment Mapping

- Main idea: fake **reflections** using simple textures

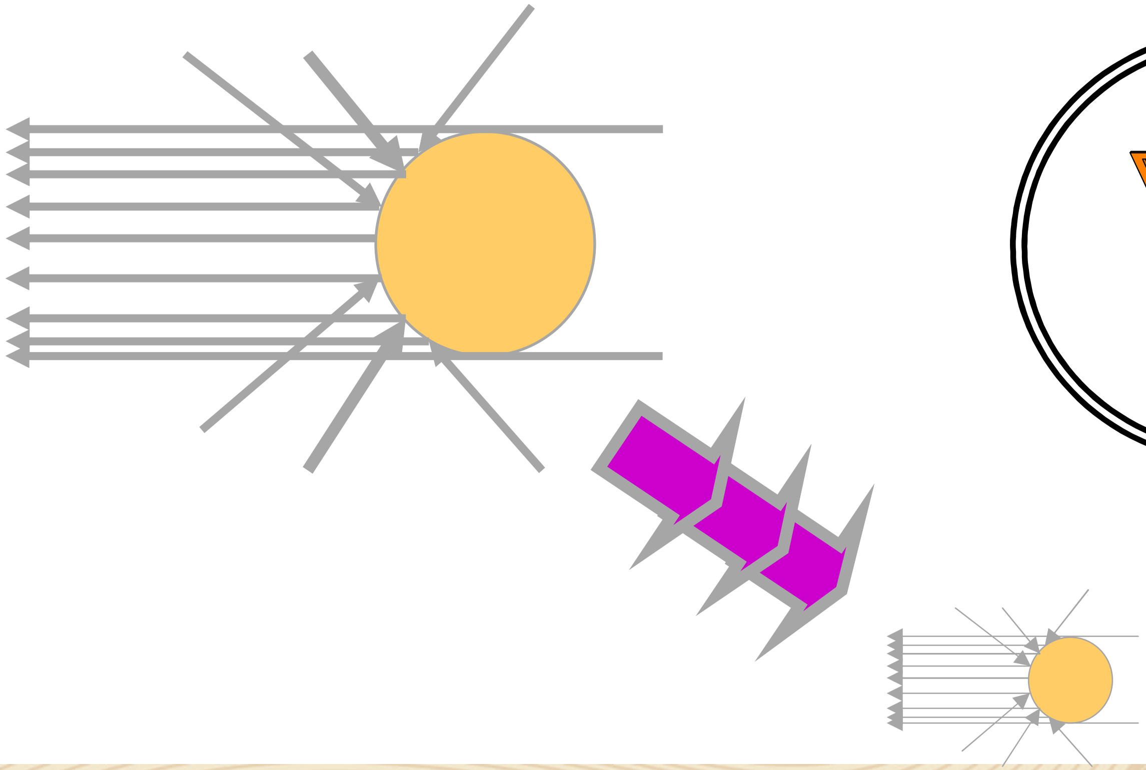
Environment Mapping

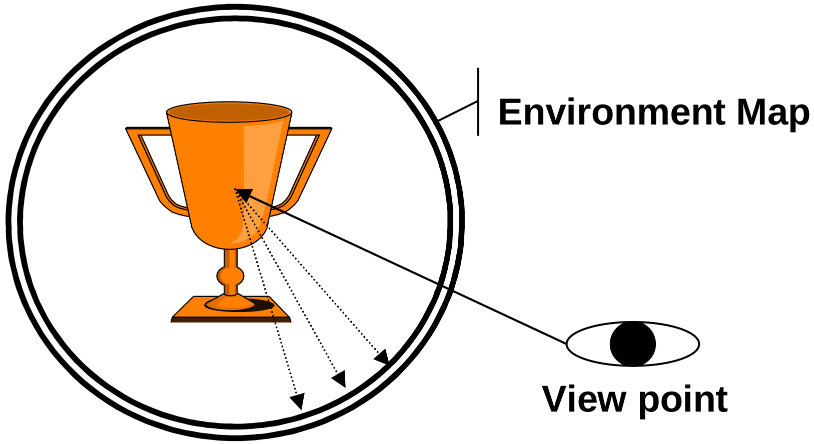
- Assumption: index env-map via **orientation**
  - Reflection vector or any other similar lookup!



Environment Mapping

- Ignore (reflection) position! True if:
  - Reflecting object shrunk to a single point
  - OR: environment infinitely far away





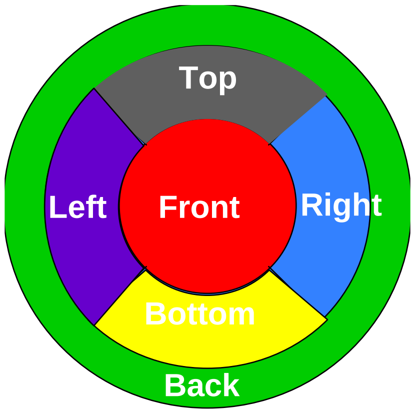
Environment Mapping

- Can be an “effect”
  - Usually means: “fake reflection”
- Can be a “technique” (i.e., GPU feature)
  - Then it means:
    “2D texture indexed by a 3D orientation”
  - Usually the index vector is the reflection vector
  - But can be anything else that’s suitable!
- Increased importance for modern GI

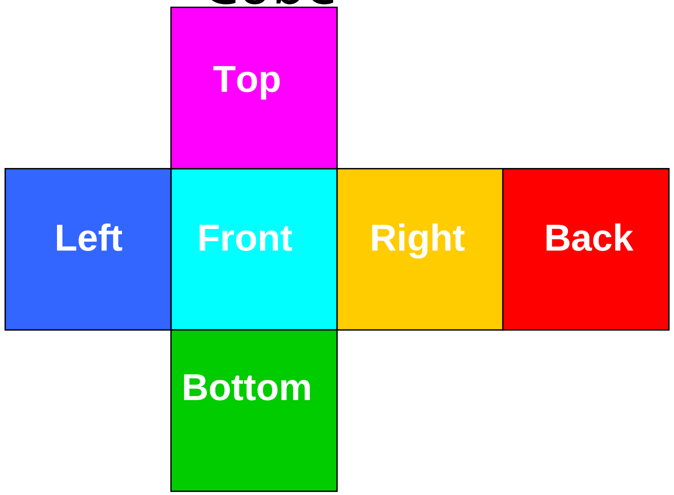
Environment Mapping

- Uses texture coordinate generation, multi-texturing, new texture targets…
- Main task
  - Map all 3D orientations to a 2D texture
- Independent of application to reflections

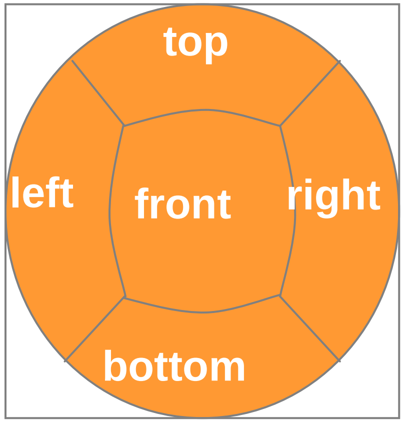
Sphere



Cube

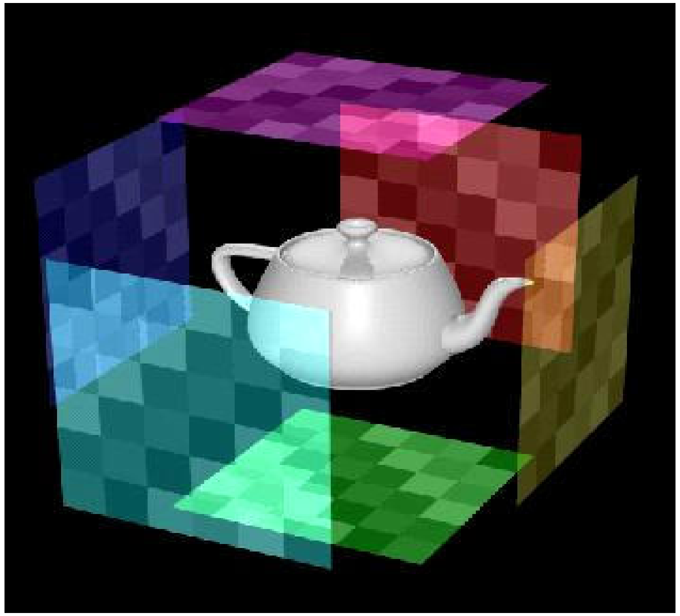


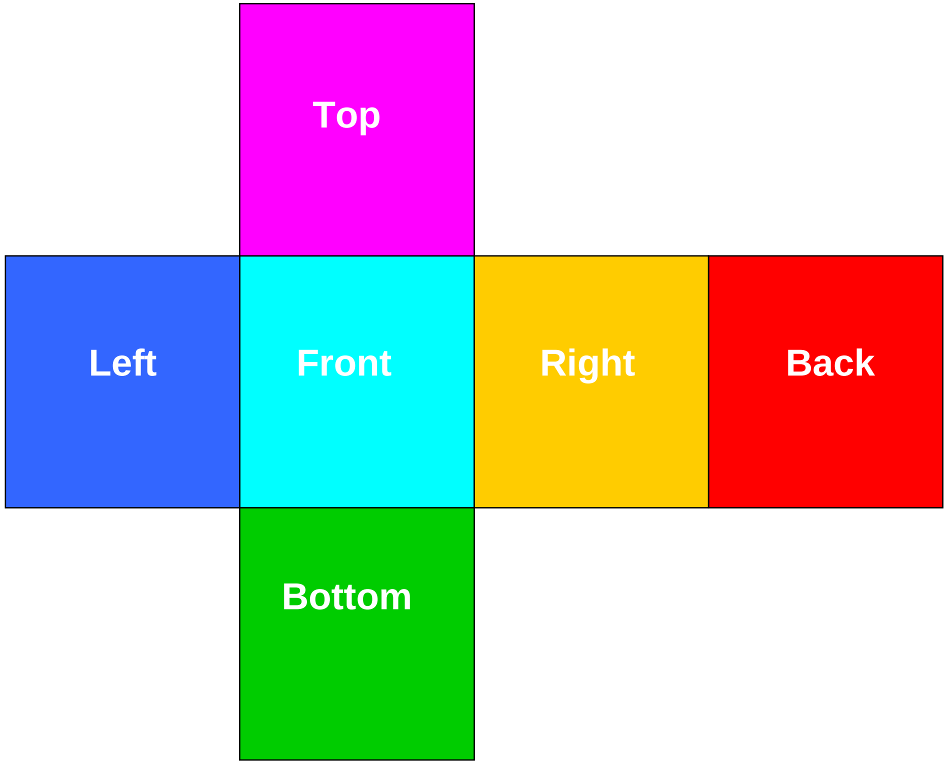
Dual paraboloid



Cube Mapping

- OpenGL texture targets

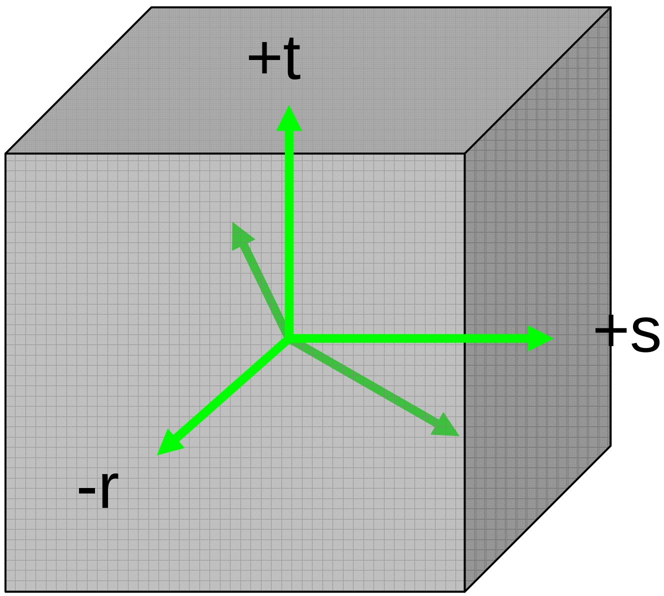





```
glTexImage2D(
GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGB8,
w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, face_px);
```

Cube Mapping

- Cube map accessed via ***vectors*** expressed as 3D texture coordinates (s, t, r)



Cube Mapping

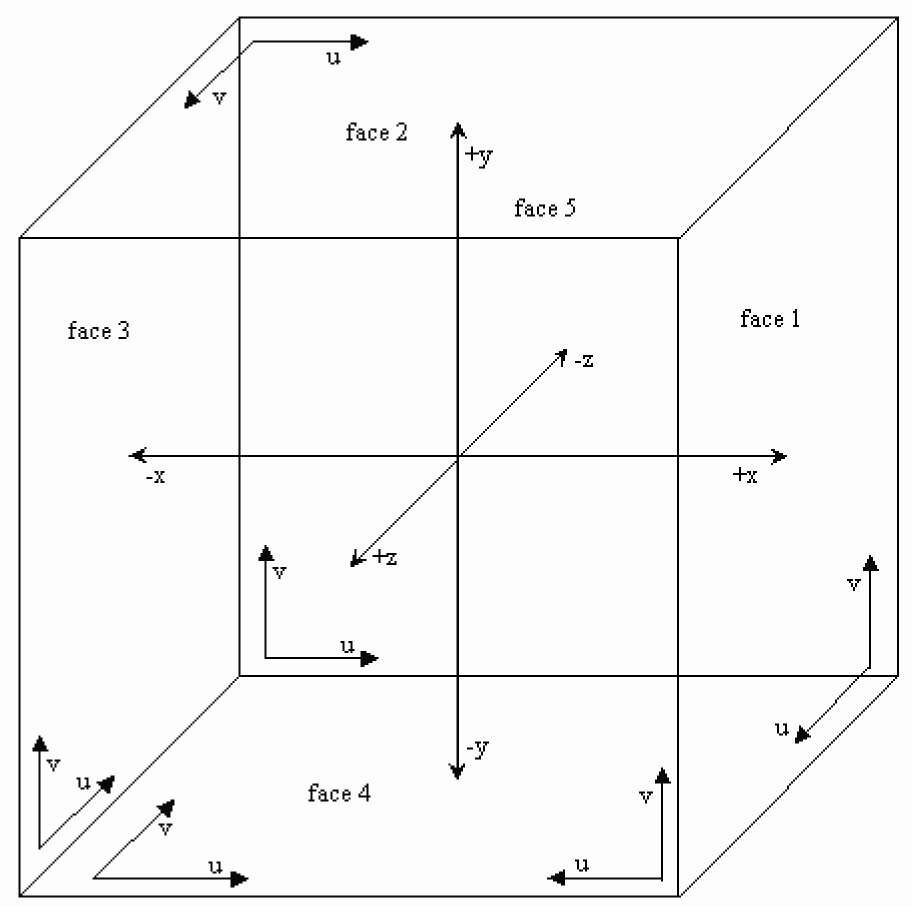
- 3D → 2D projection done by hardware
  - Highest magnitude component selects which cube face to use (e.g., -t)
  - Divide other components by this, e.g.:
    $s' = s / -t$
    $r' = r / -t$
  - (s', r') is in the range $[-1, 1]$
  - remap to $[0,1]$ and select a texel from selected face

- Still need to *generate* useful texture coordinates for reflections

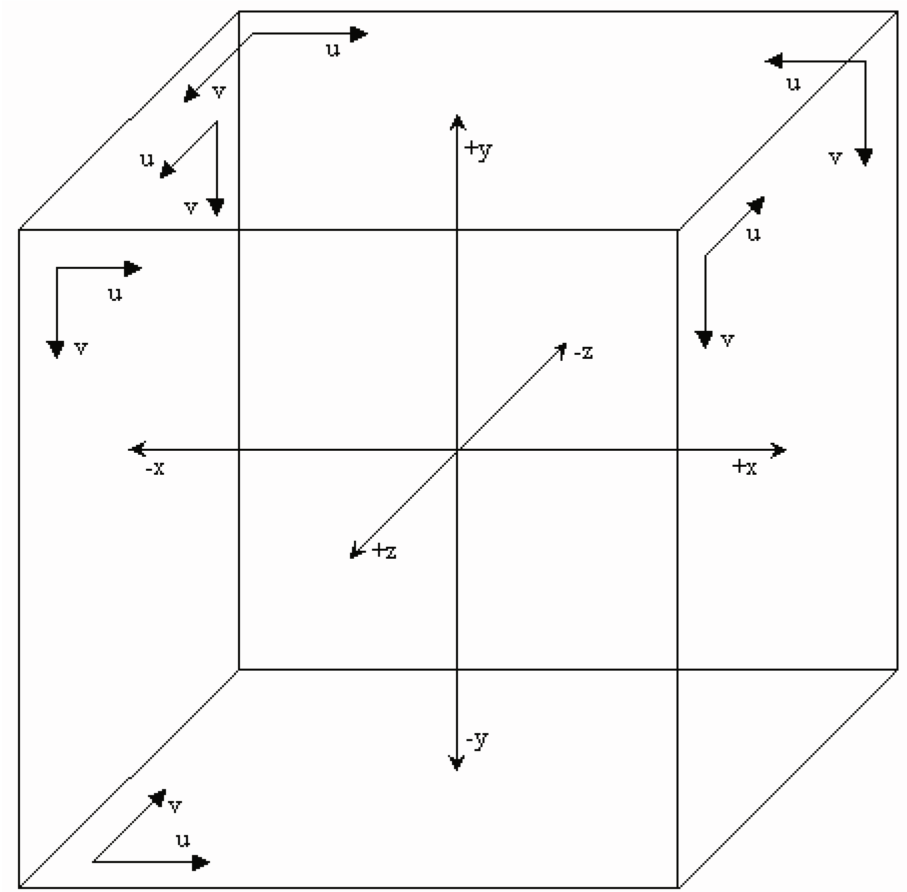
Cube Mapping

- Generate views of the environment
  - One for each cube face
  - 90° view frustum
  - Use hardware render to texture
  - `textureCube(samplerCube, vec3 dir);`
  - `textureLod(samplerCube, vec3 dir, level);`

Cube Map Coordinates



Watt 3D CG



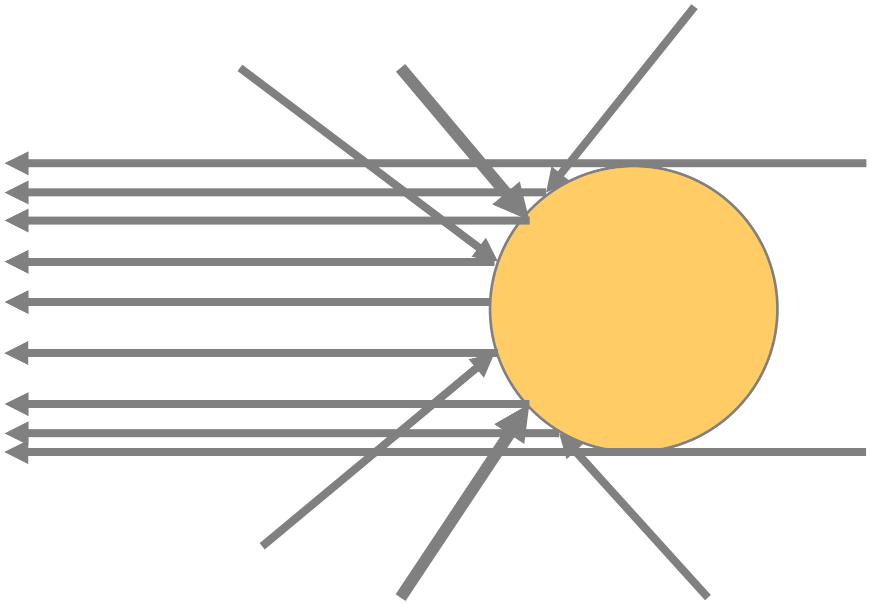
Renderman/OpenGL

Cube Mapping

- Advantages
  - Minimal distortions
  - Creation and map entirely hardware accelerated
  - Can be generated dynamically
- Optimizations for dynamic scenes
  - Need not be updated every frame
  - Low resolution sufficient

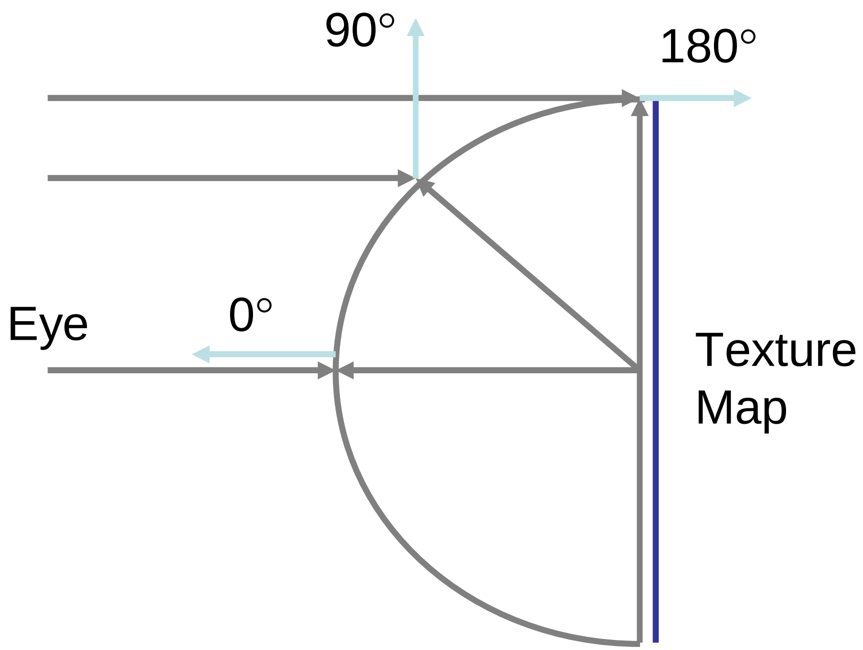
Sphere Mapping

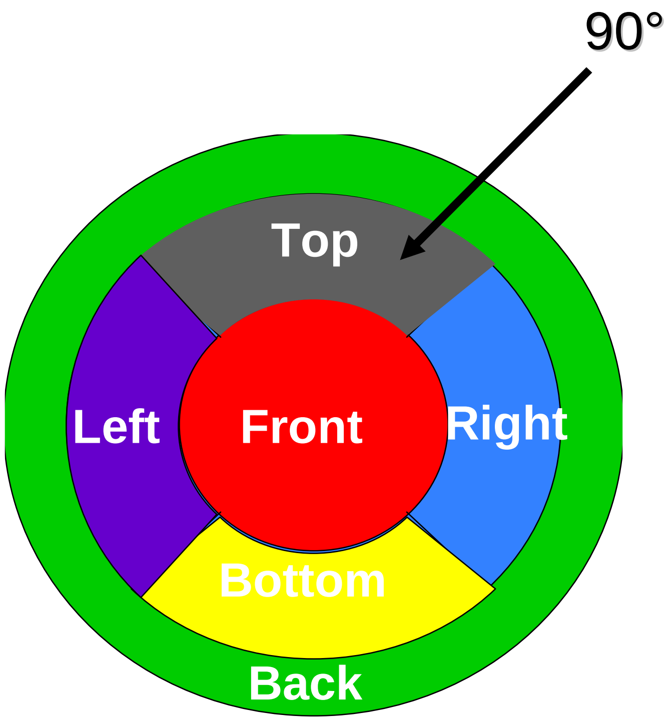
- Earliest available method with OpenGL
  - Only texture mapping required!
- Texture looks like ***orthographic*** reflection from chrome hemisphere
  - Can be photographed like this!



Sphere Mapping

- Maps all reflections to hemisphere
  - Center of map reflects back to eye
  - Singularity: back of sphere maps to outer ring



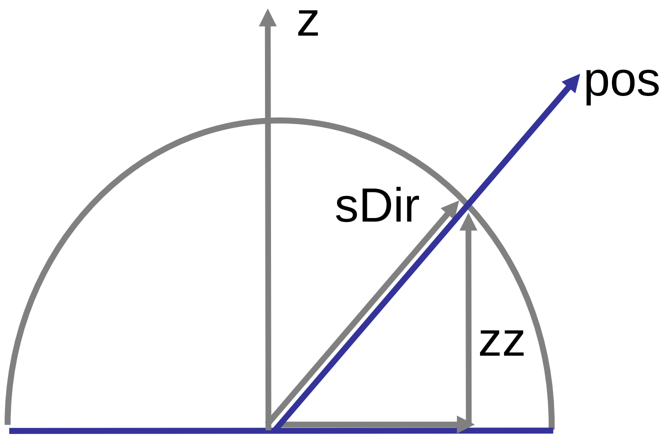


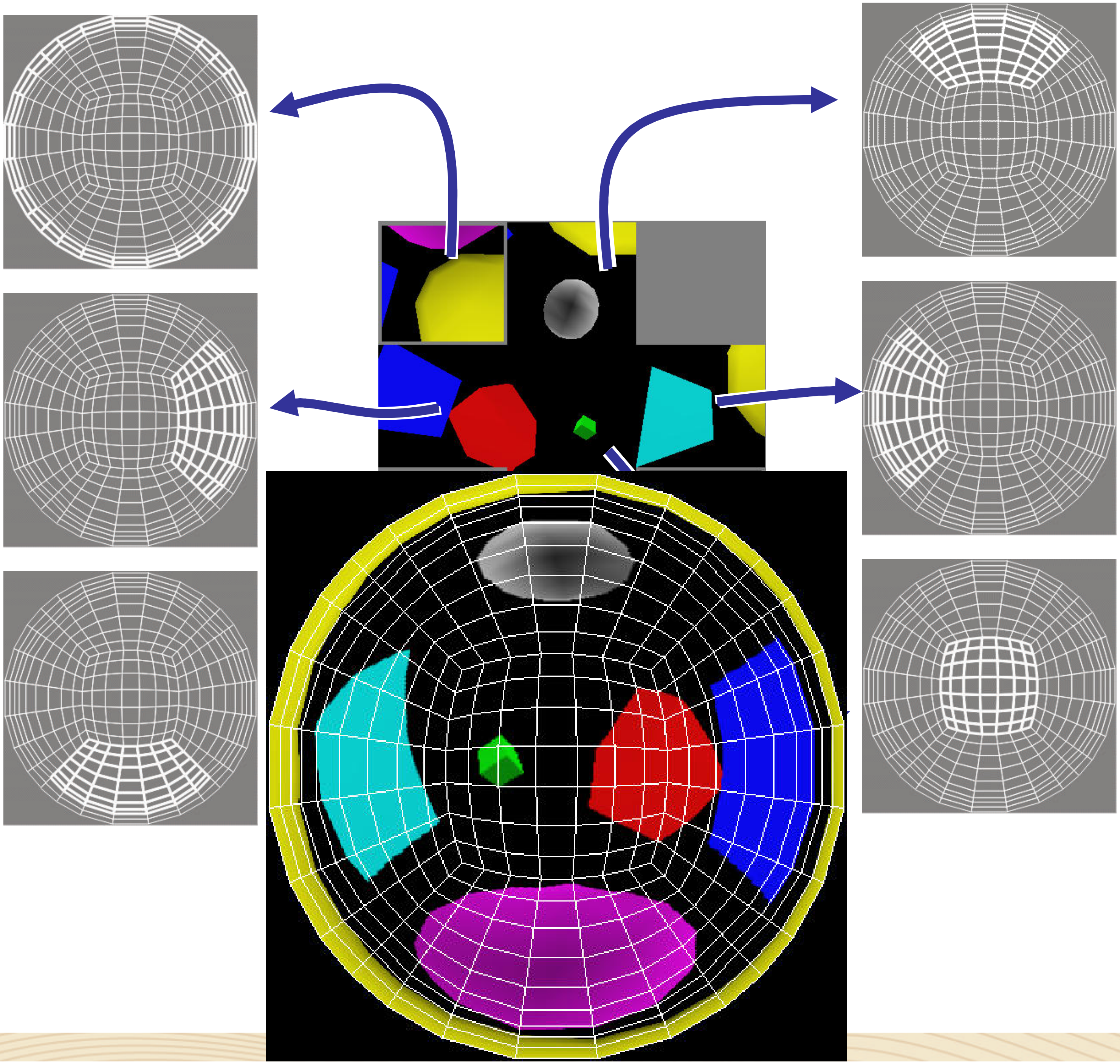
Rasterizing None Linear Mappings

- Linear interpolation does not work anymore
  - Avoid long edges
- Approximate by subdividing big triangles
- Problems at horizon due to straddeling triangles

Sphere Mapping

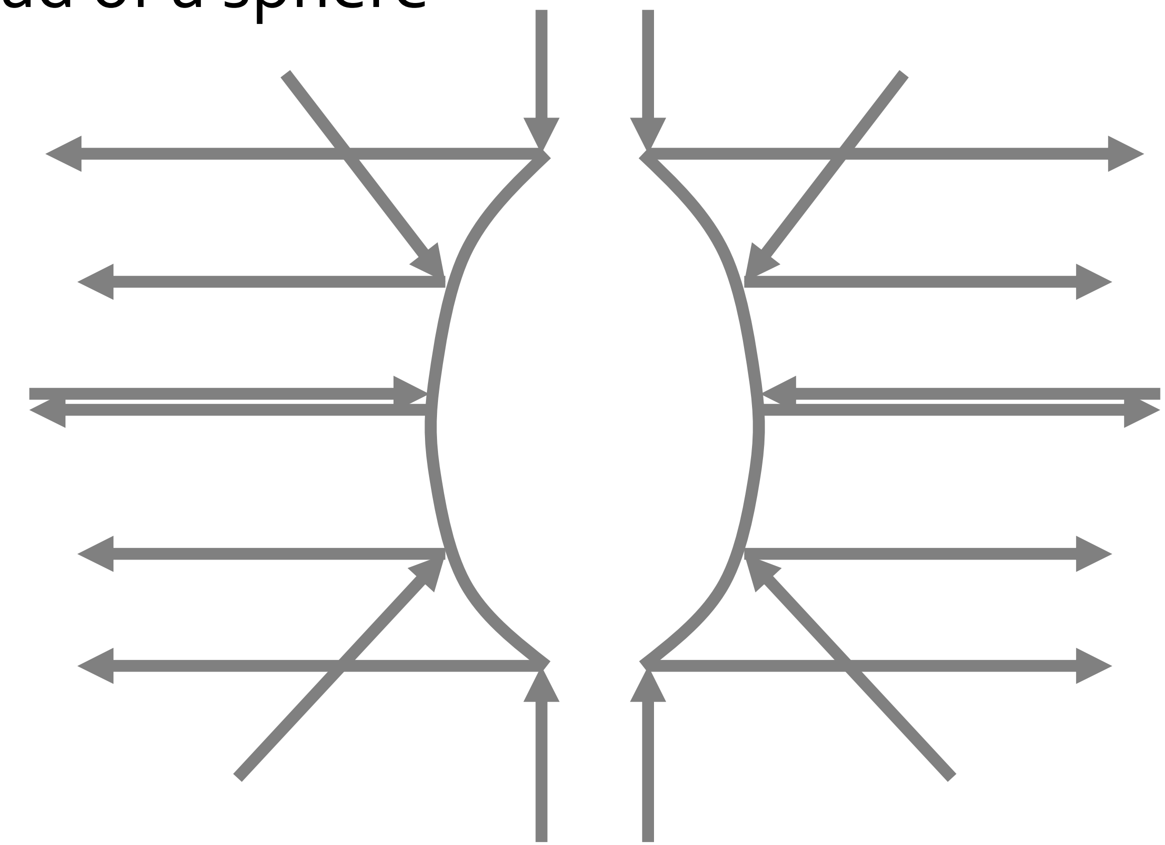
- Projection onto unit sphere
```
normalize(vec3 pos).xy
```
- Back from sphere:
```
vec3 unproject(vec2 sDir) {
   float zz = 1 – dot(sDir, sDir);
   return vec3(sDir.x, sDir.y, sqrt(zz));}
```





(Dual) Paraboloid Mapping

- Use orthographic reflection of two parabolic mirrors instead of a sphere



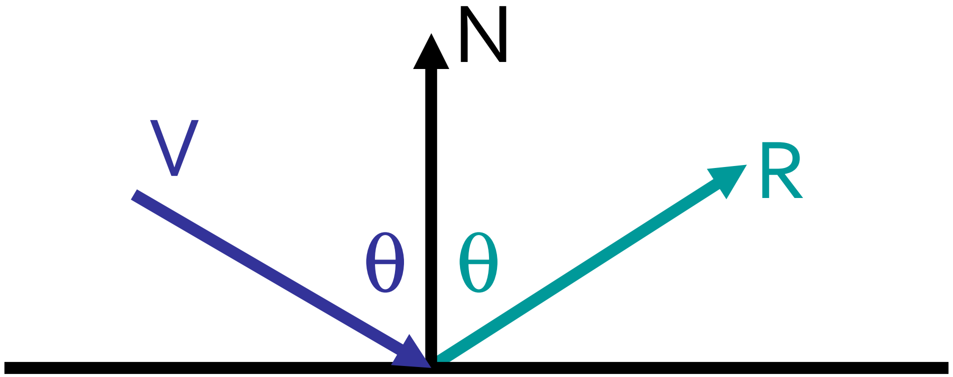
(Dual) Paraboloid Mapping

- Projection onto parabola
  `pos.xy / (pos.z - 1)`
- Back from parabola:

```
vec3 unproject(vec2 sDir) {
   float z = 0.5 – 0.5 * dot(sDir, sDir);
   return vec3(sDir.x, sDir.y, z);}
```

Reflective Environment Mapping

- Angle of incidence = angle of reflection



R = V - 2 (N dot V) N
= `reflect(V,N)`

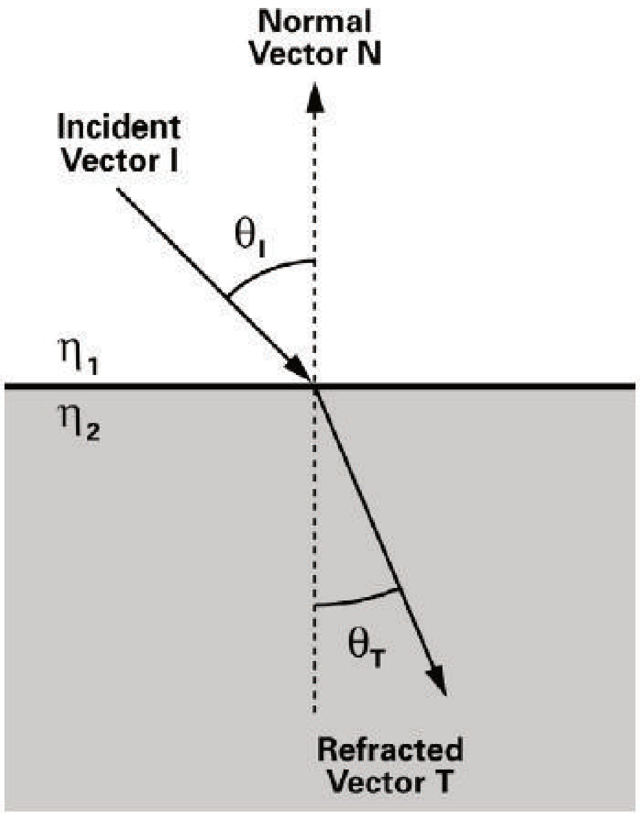
V and N normalized!

V is incident vector!

- Cube map needs reflection vector in coordinates (where map was created)

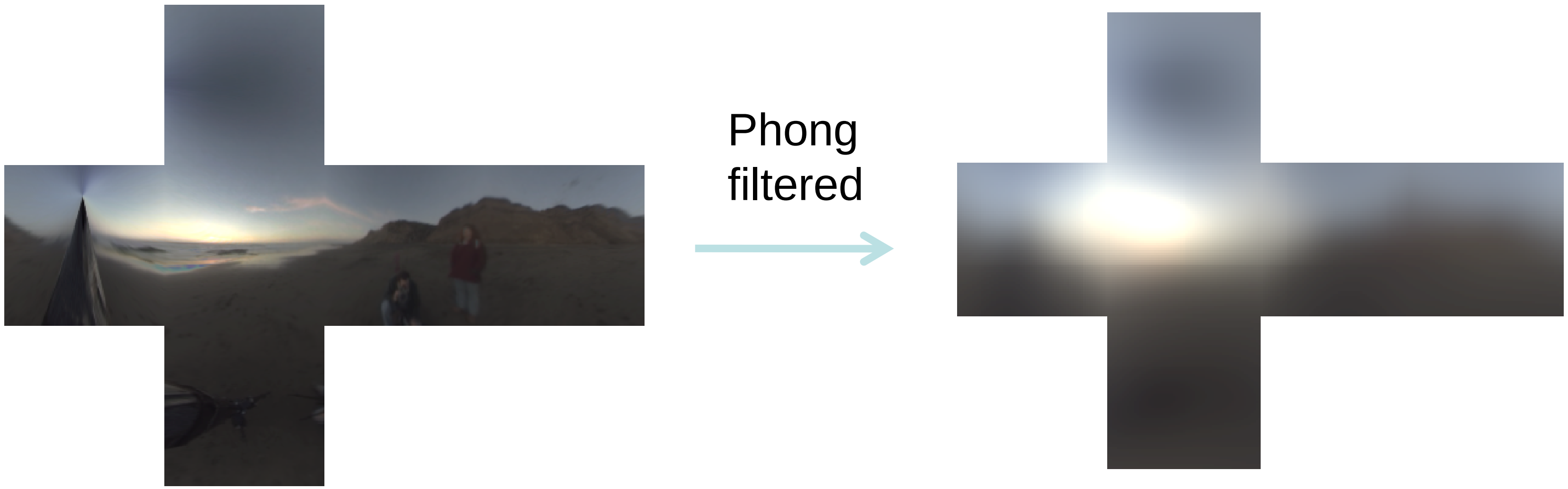
Refractive Environment Mapping

- Use refracted vector for lookup:
  - Snells law: $\eta_1 \sin\theta_I = \eta_2 \sin\theta_T$



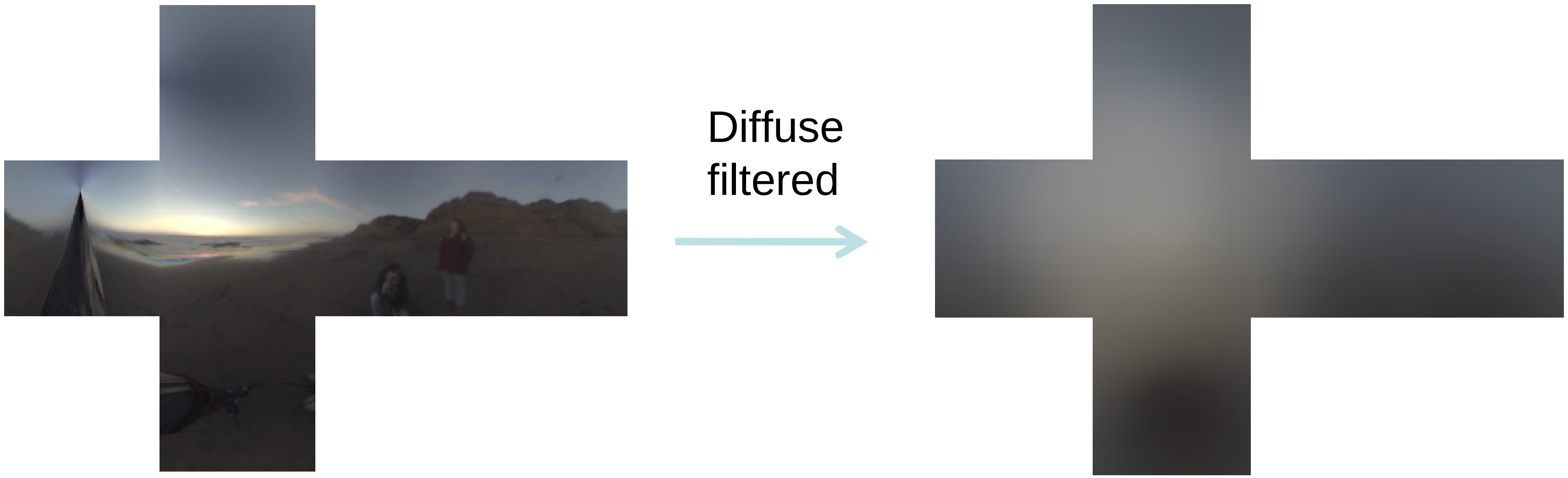
Specular Environment Mapping

- We can pre-filter the environment map
  - Equals specular integration over the hemisphere
  - Phong lobe (cos^n) as filter kernel
  - **`textureLod`** with **`level`** according to glossiness
  - R as lookup



Irradiance Environment Mapping

- Pre-filter with cos (depends on mapping)
  - Lambert cos already integrated
  - Paraboloid not integrated
  - Equals diffuse integral over hemisphere
  - N as lookup direction



Environment Mapping Conclusions

- "Cheap" technique
  - Highly effective for static lighting
  - Simple form of image based lighting
    - Expensive operations are replaced by pre-filtering
- Advanced variations:
  - Separable BRDFs for complex materials
  - Real-time filtering of environment maps
  - Fresnel term modulations (water, glass)
- Used in virtually every modern computer game

Environment Mapping Toolset

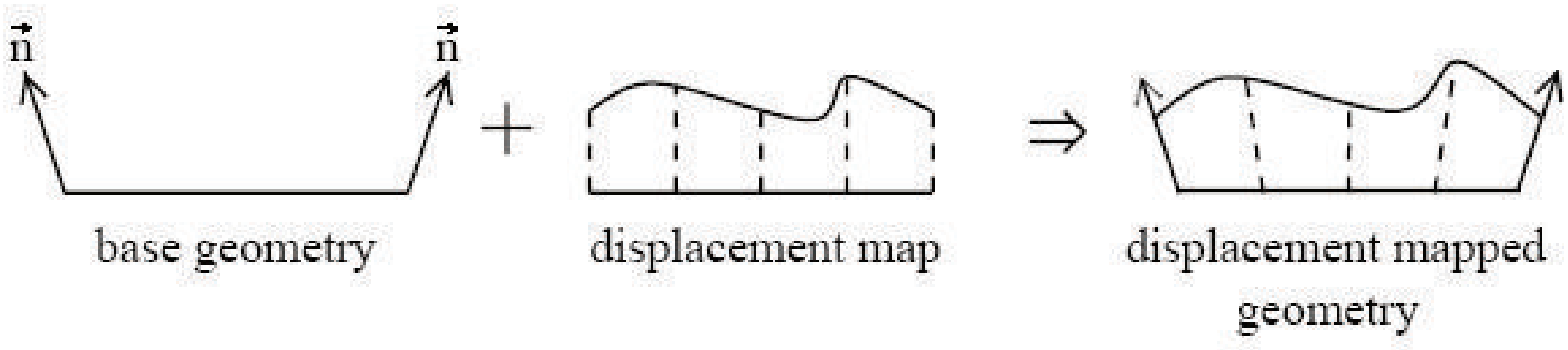
- Environment map creation:
  - AMDs CubeMapGen (free)
    - Assembly
    - Proper filtering
    - Proper MIP map generation
    - Available as library for your engine/dynamic environment maps
  - HDRShop 1.0 (free)
    - Representation conversion
      - Spheremap to Cubemap

Advanced Texturing

Displacement Mapping

Displacement Mapping

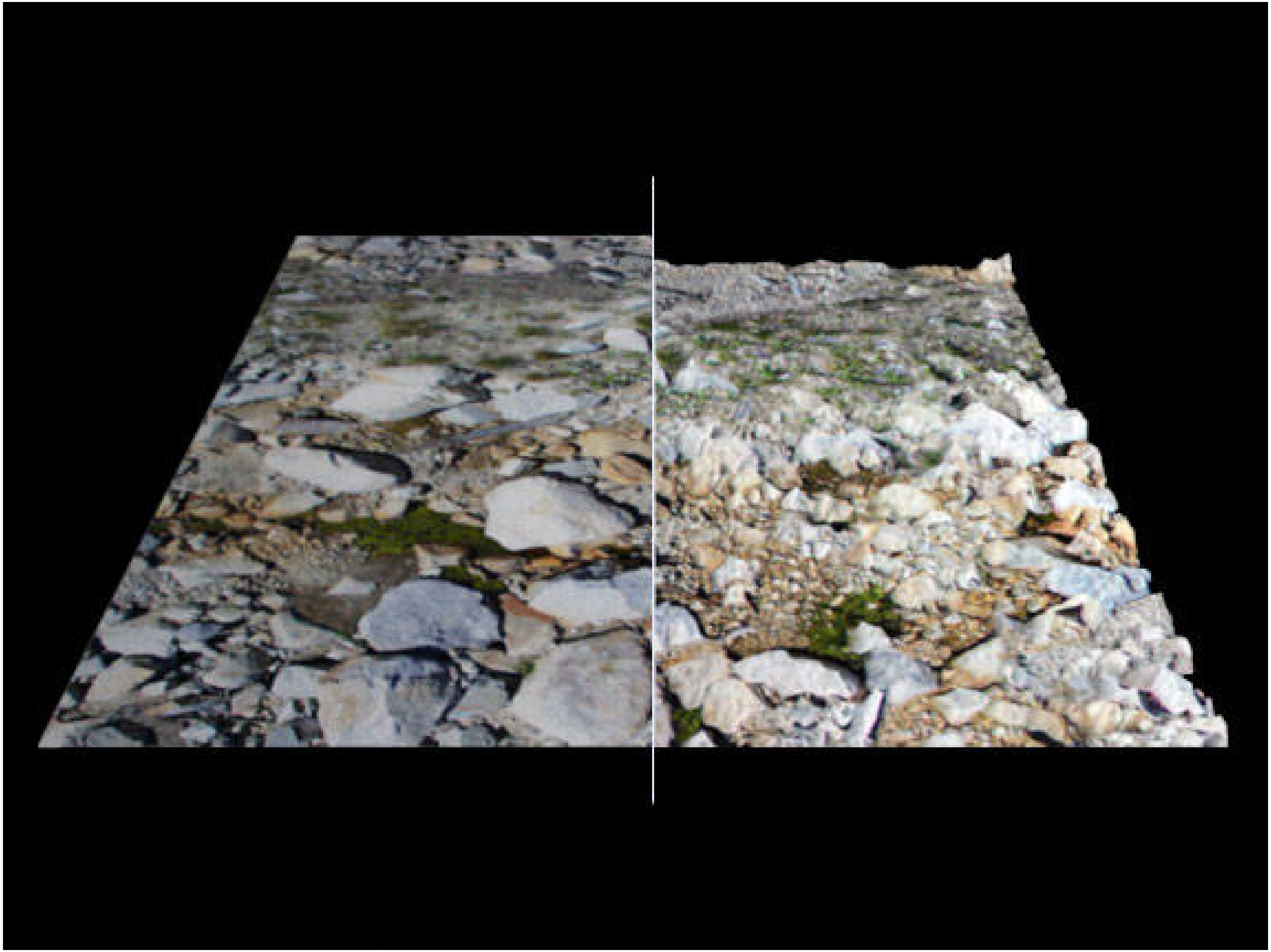
- A displacement map specifies displacement in the direction of the surface normal, for each point on a surface



Idea

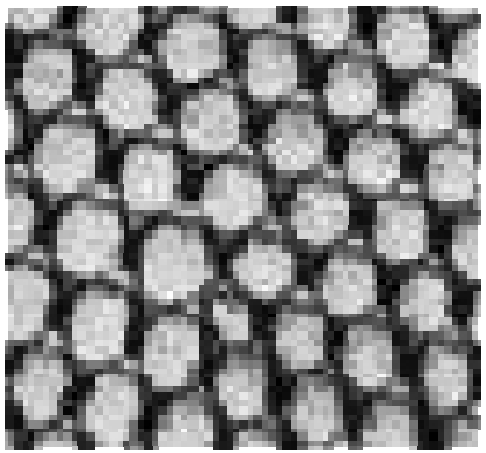
- Displacement mapping shifts all points on the surface in or out along their normal vectors
- Assuming a displacement texture d,

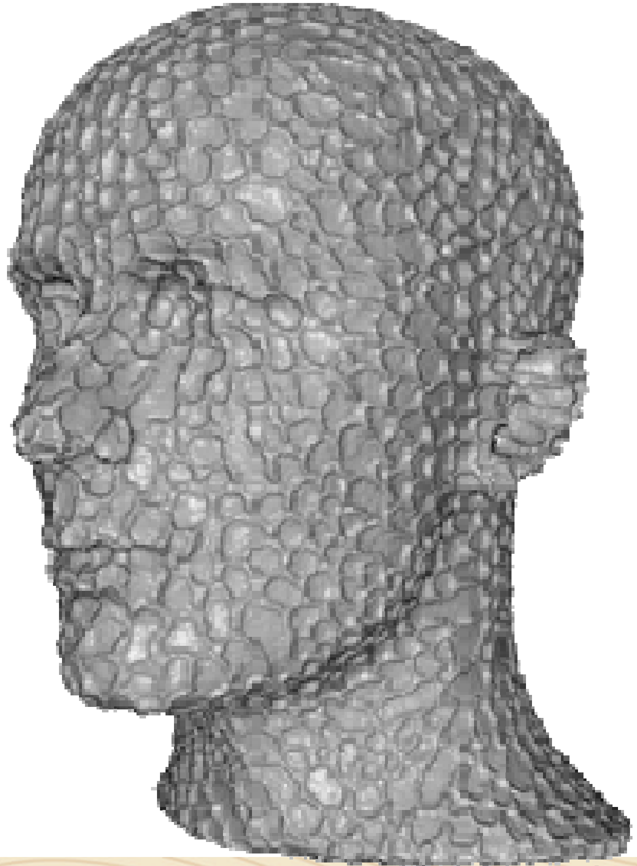
  $p' = p + d(p) * n$



Displacement Map

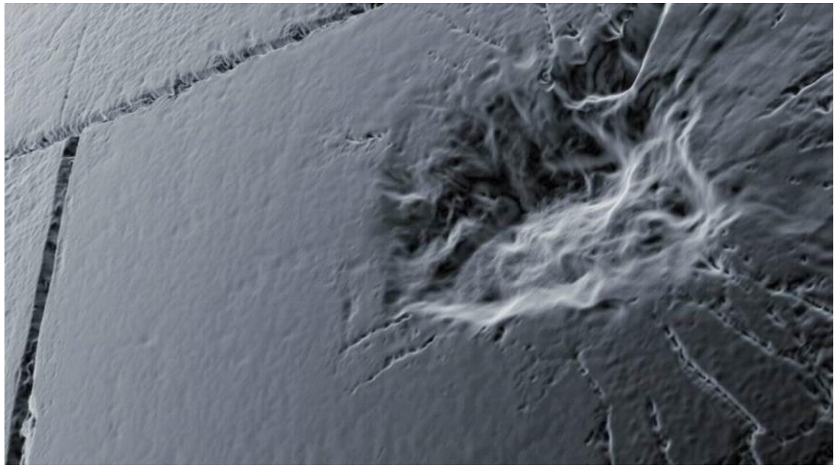
- Store only geometric details
- Not a parameterization (from subdivision surface)
- Just a scalar-valued function.





Displacement Mapping

- Function of u,v texture coordinates (or parametric surface parameters)
- Stored as a 2d texture
- And/or computed procedurally
- Problem:
  How can we render a model given as a set of polygons, and a displacement map?



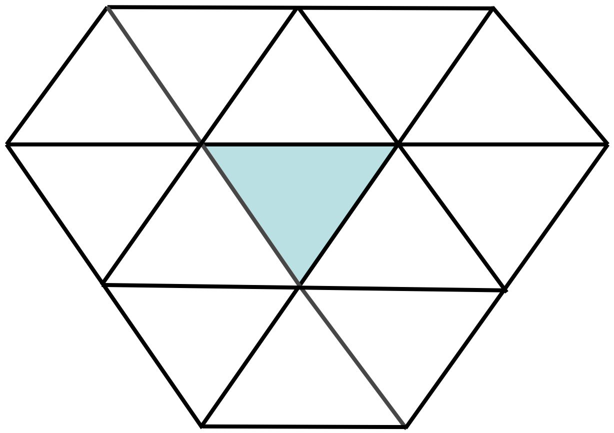
Approaches

- Geometric
  - Subdivide and displace
  - Volume slice rendering
  - Ray tracing
  - Tessellation HW
- Image Space
  - Parallax mapping
  - Relief textures
  - View dependent texturing / BDTF
  - View dependent displacement mapping

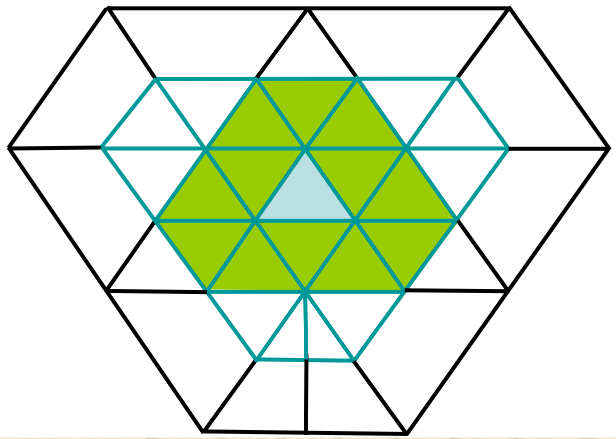
Subdivide and Displace

- Subdivide each polygon
- Displace each vertex along normal using displacement map
- Many new vertices and triangles
- All need to be transformed and rendered
- Improvements
  - Adaptive subdivision
  - Hardware implementation

Regular patch

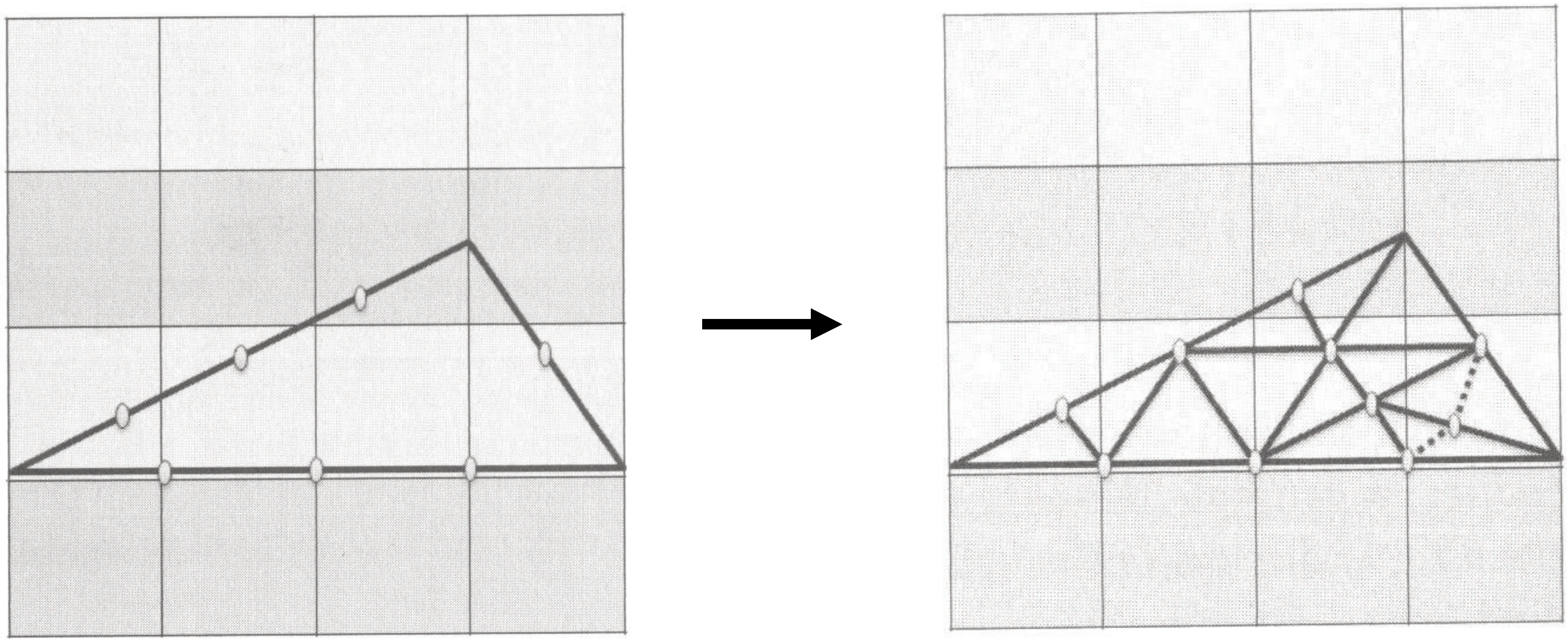


Irregular patch
after one level of subdivision



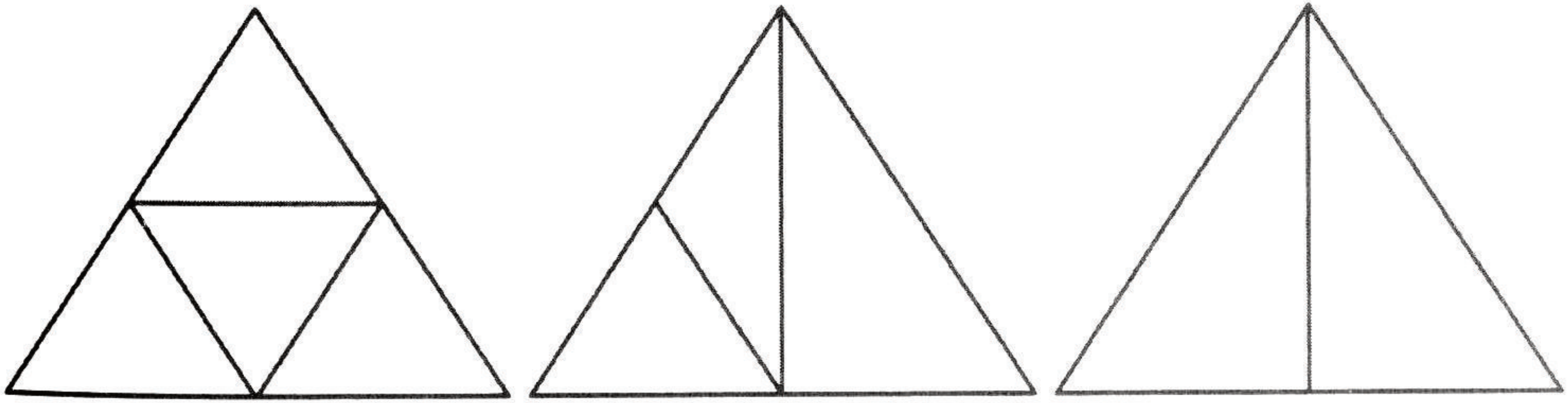
Simple Adaptive Subdivision

- 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



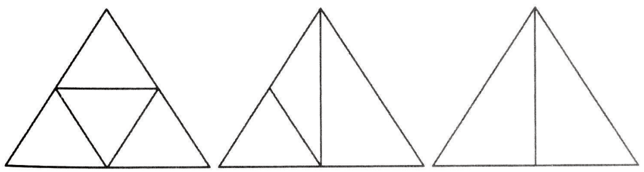
3 edge split

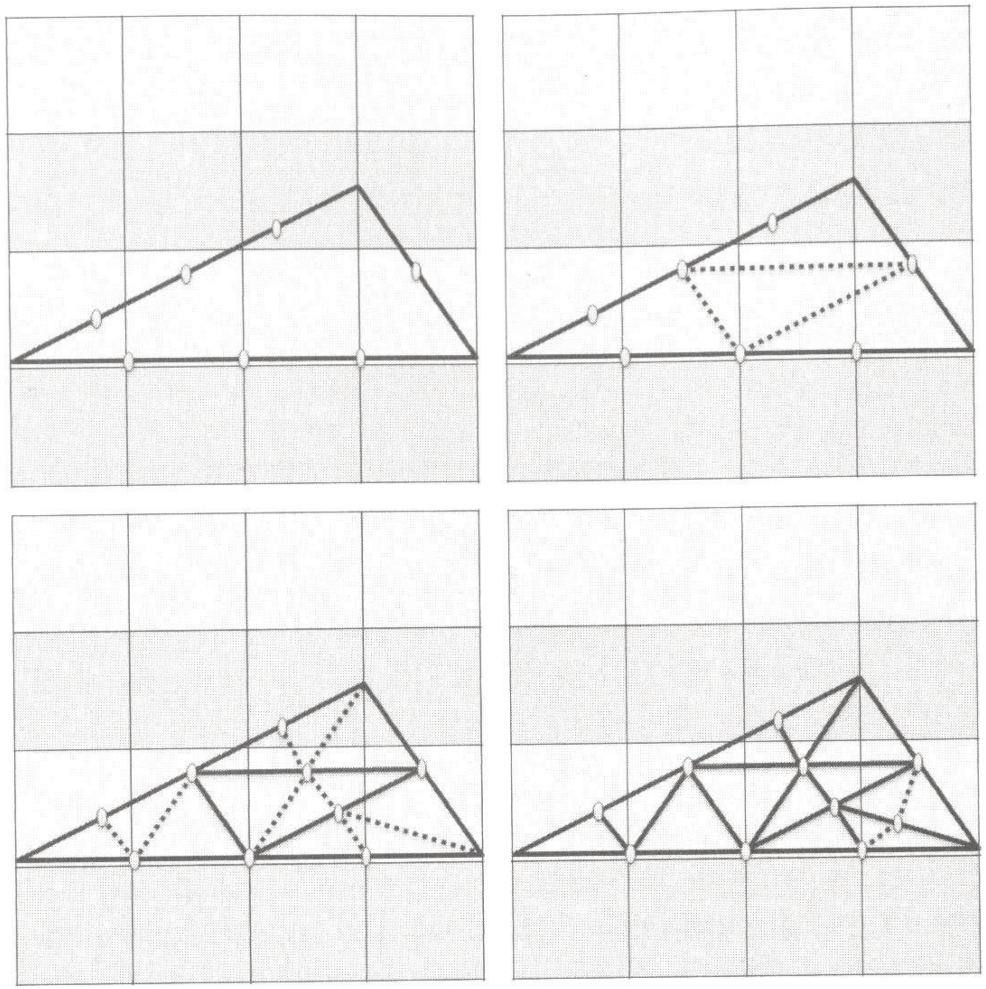
2 edge split

1 edge split

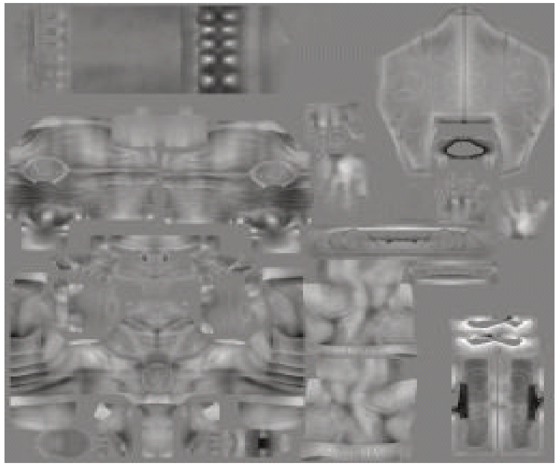
Simple Adaptive Subdivision

- Precomputed tessellation patterns
- Recursive subdivision





Displaced Subdivision



Advanced Texturing

Normal (Bump) Mapping

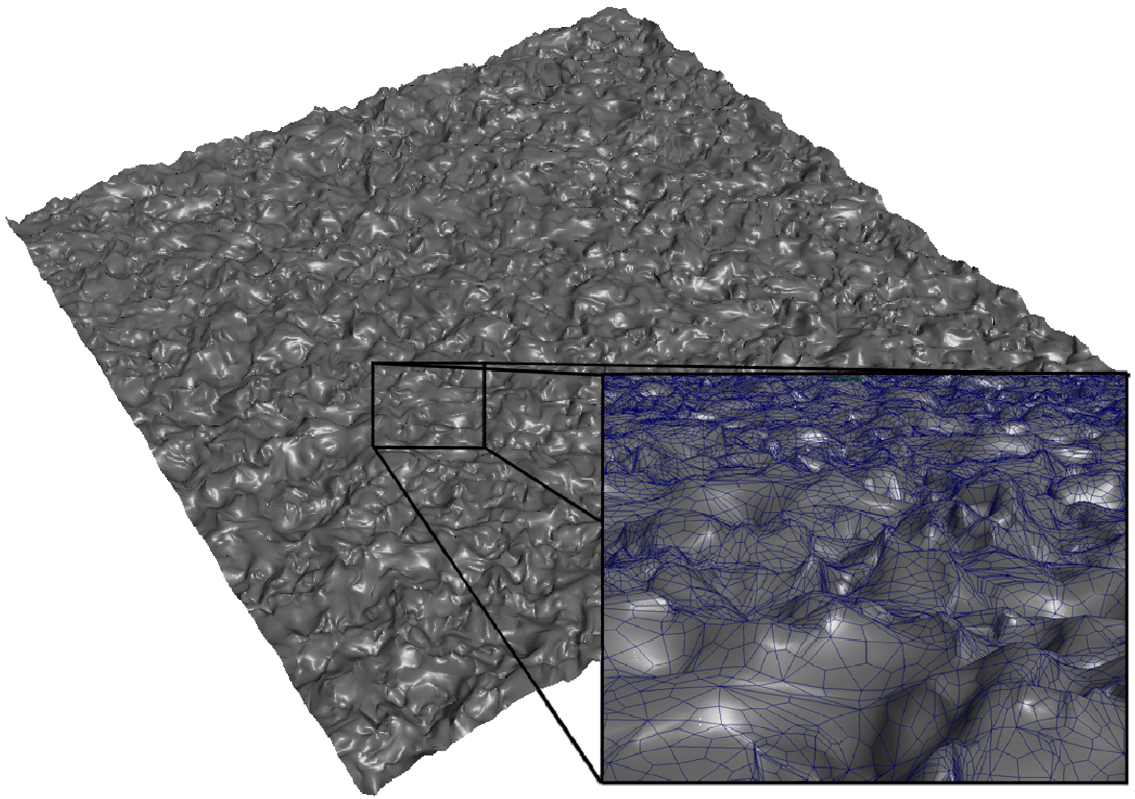
Normal Mapping

- 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

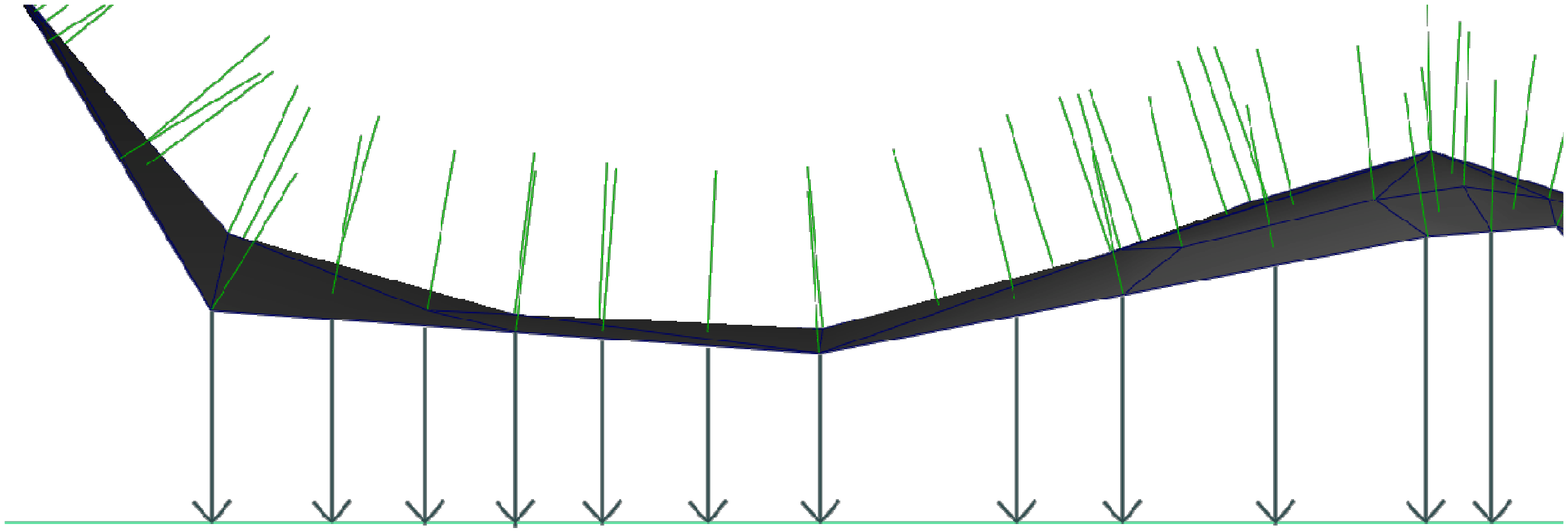
Normal Mapping

- Fine structures require a massive amount of polygons
- Too slow for full scene rendering



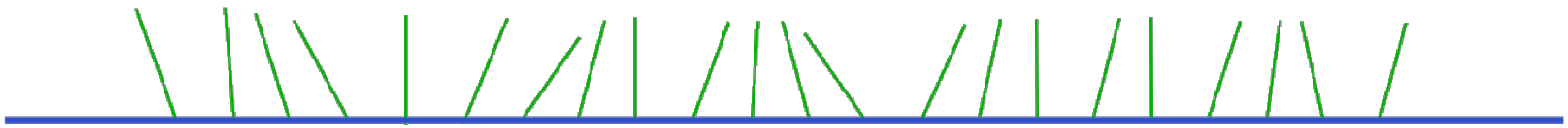
Normal Mapping

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



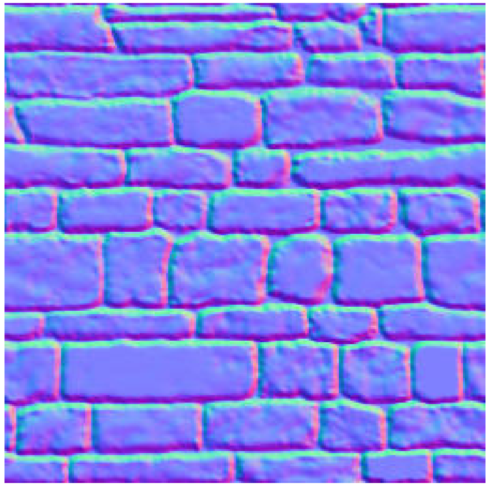
Normal Mapping

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



Normal Mapping

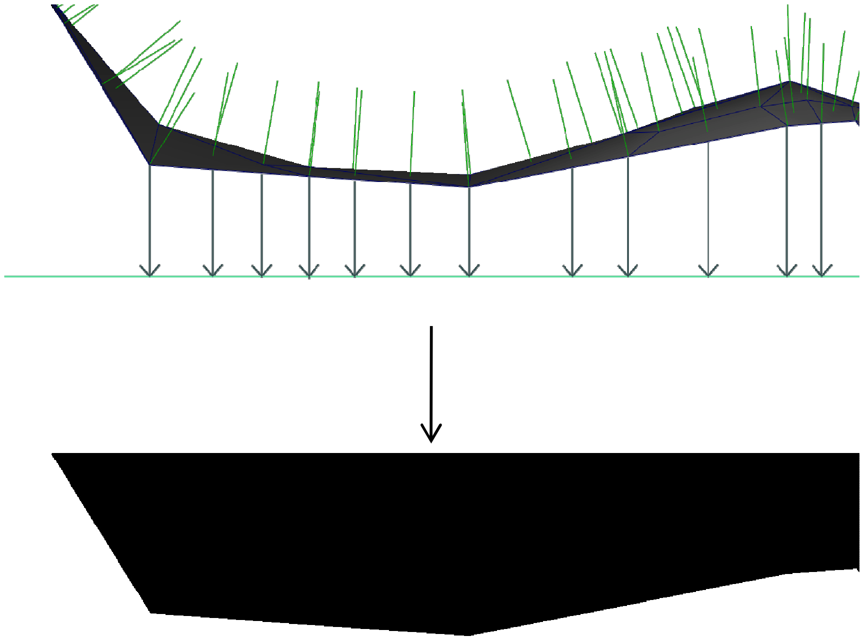
- Result: Texture that contains the normals as vectors
  - Red X
  - Green Y
  - Blue Z
  - Saved as range compressed bitmap ([-1..1] mapped to [0..1])



- Directions instead of polygons!
- Shading evaluations executed with lookup normals instead of interpolated normal

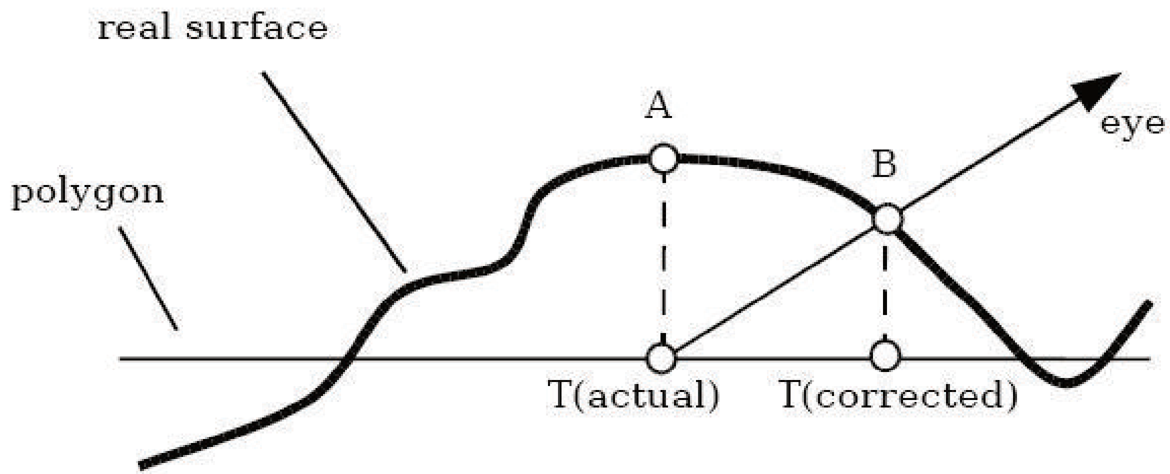
Normal Mapping

- Additional result is height field texture
  - Encodes the distance of original geometry to the carrier geometry



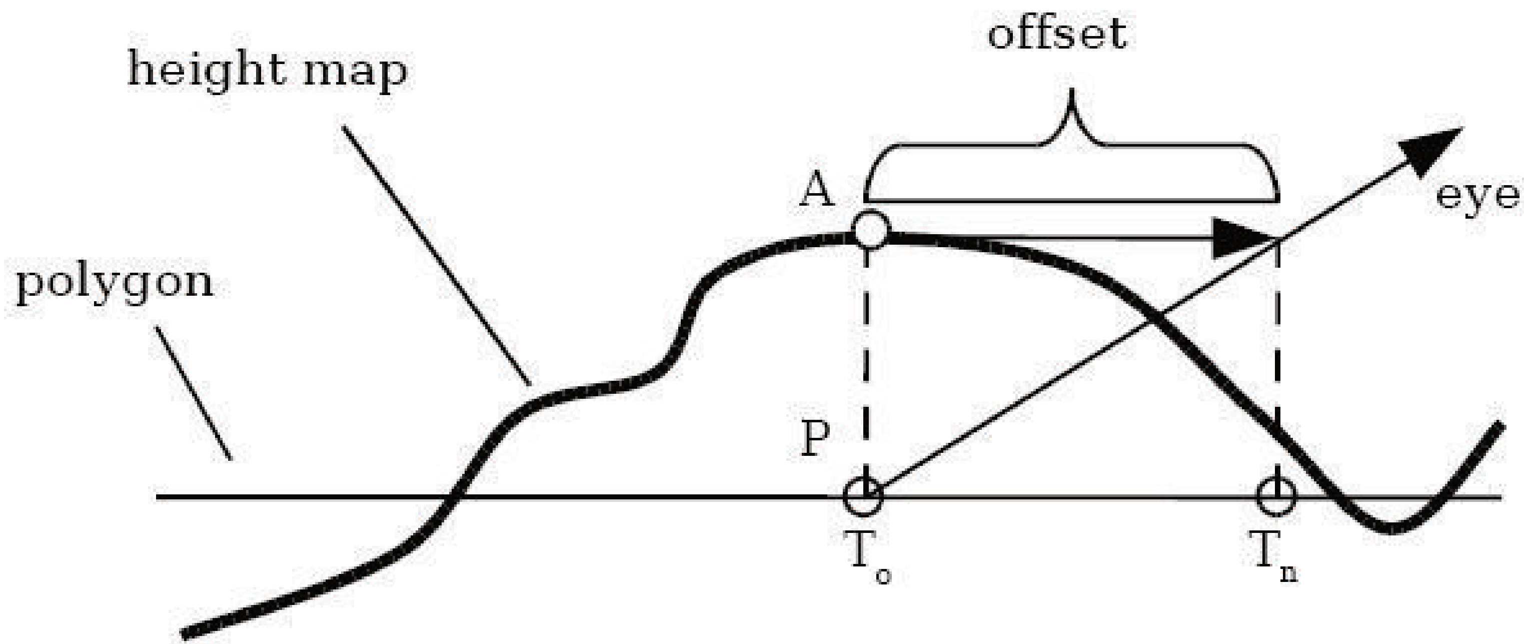
Parallax-Normal Mapping

- 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



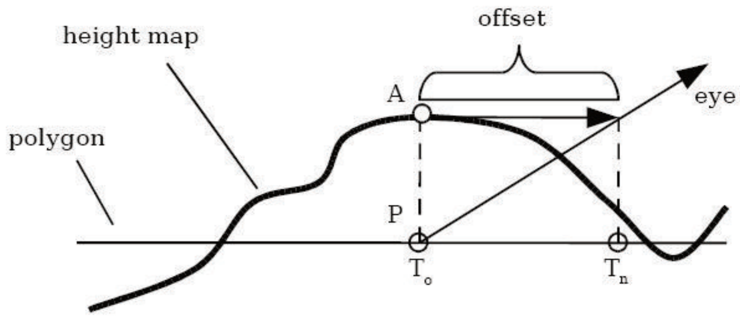
Parallax-Normal Mapping

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



Parallax-Normal Mapping

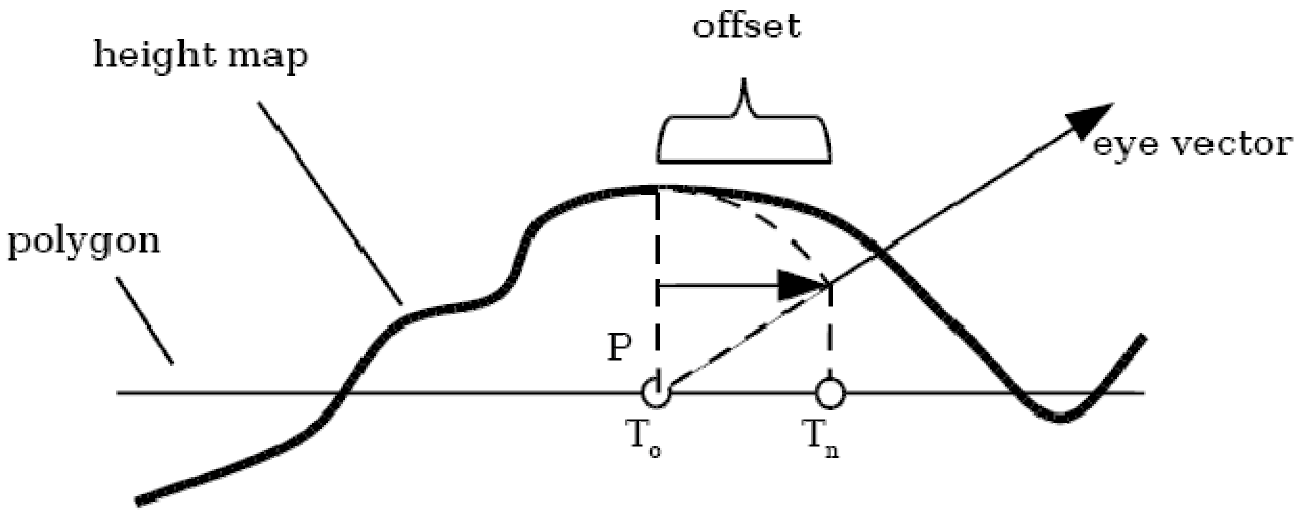
- Rescale height map h to appropriate values:
  $h_n = h*s - 0.5s$
  (s = scale = 0.01)



- Assume height field is locally constant
  - Lookup height field at T_o
- Trace ray from T_o to eye with eye vector V to height and add offset:
  - $T_n = T_o + (h_n * V_{x,y}/V_z)$

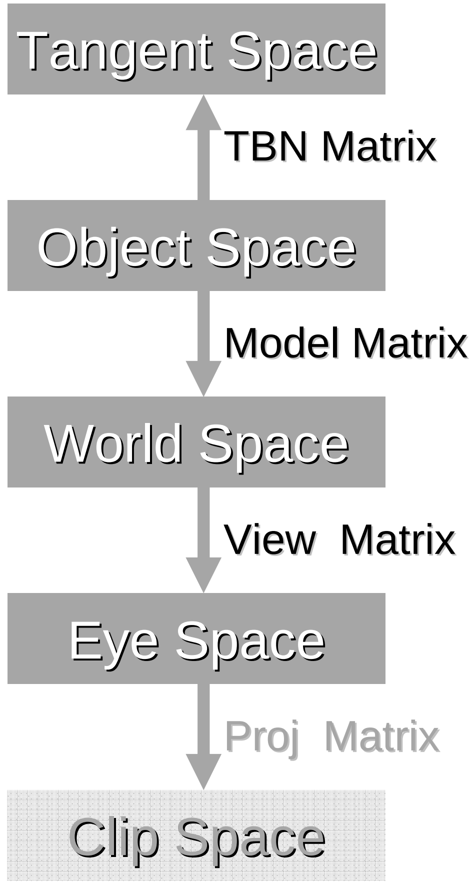
Offset Limited Parallax-Normal Mapping

- Problem: At steep viewing angles, V_z goes to zero
  - Offset values approach infinity
  - Solution: we leave out V_z division:
    $T_n = T_o + (h_n * V_{x,y})$
- Effect: offset is limited



Coordinate Systems

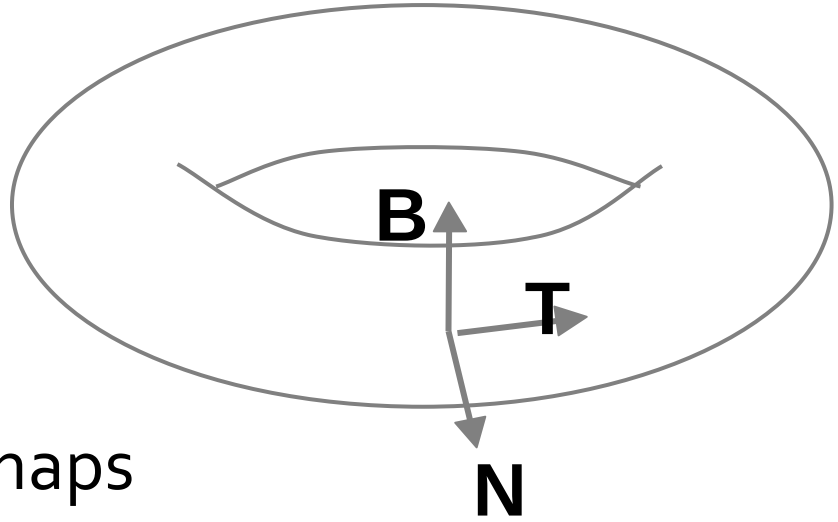
- Problem: where to calculate lighting?
- Object coordinates
  - Native space for normals (N)
- World coordinates
  - Native space for light vector (L), env-maps
  - Not explicit in OpenGL!
- Eye Coordinates
  - Native space for view vector (V)
- Tangent Space
  - Native space for normal maps



Tangent Space

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

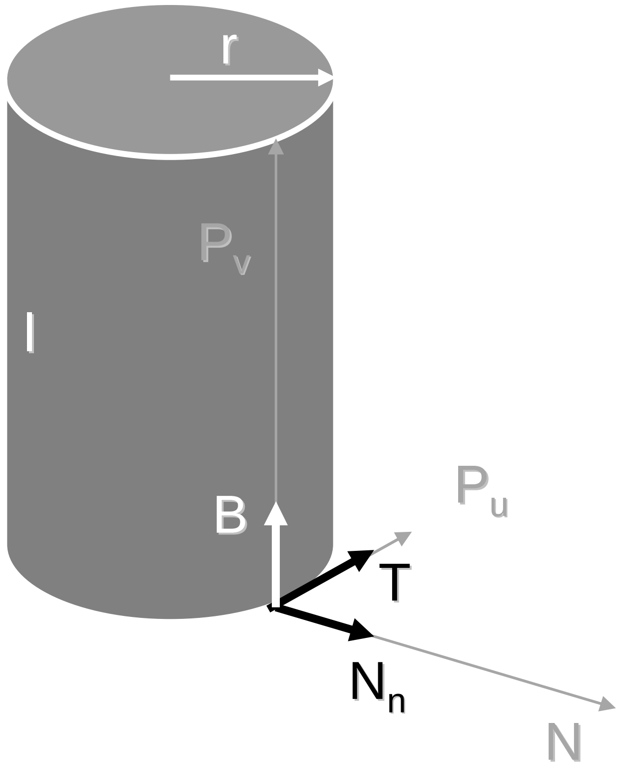
  $N_n(u,v) = P_u \times P_v \,/\, |P_u \times P_v|$
  $T = P_u \,/\, |P_u|$
  $B = N_n \times T$



- 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)

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

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
- Get vertex tangents by averaging tangents of all triangles sharing the vertex

Creating Tangent Space for a Mesh

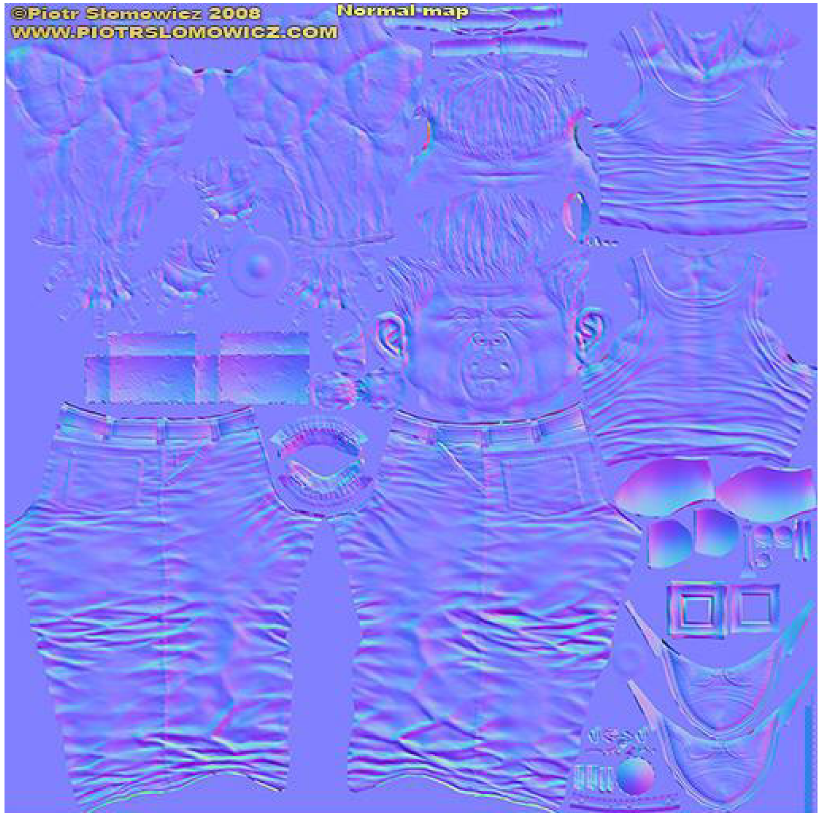
- Need to orthogonalize resulting tangents for inverting with transpose
  - Gram-Schmidt
- If only one tangent is stored we need to account for handedness.
- Code and details at http://www.terathon.com/code/tangent.html

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

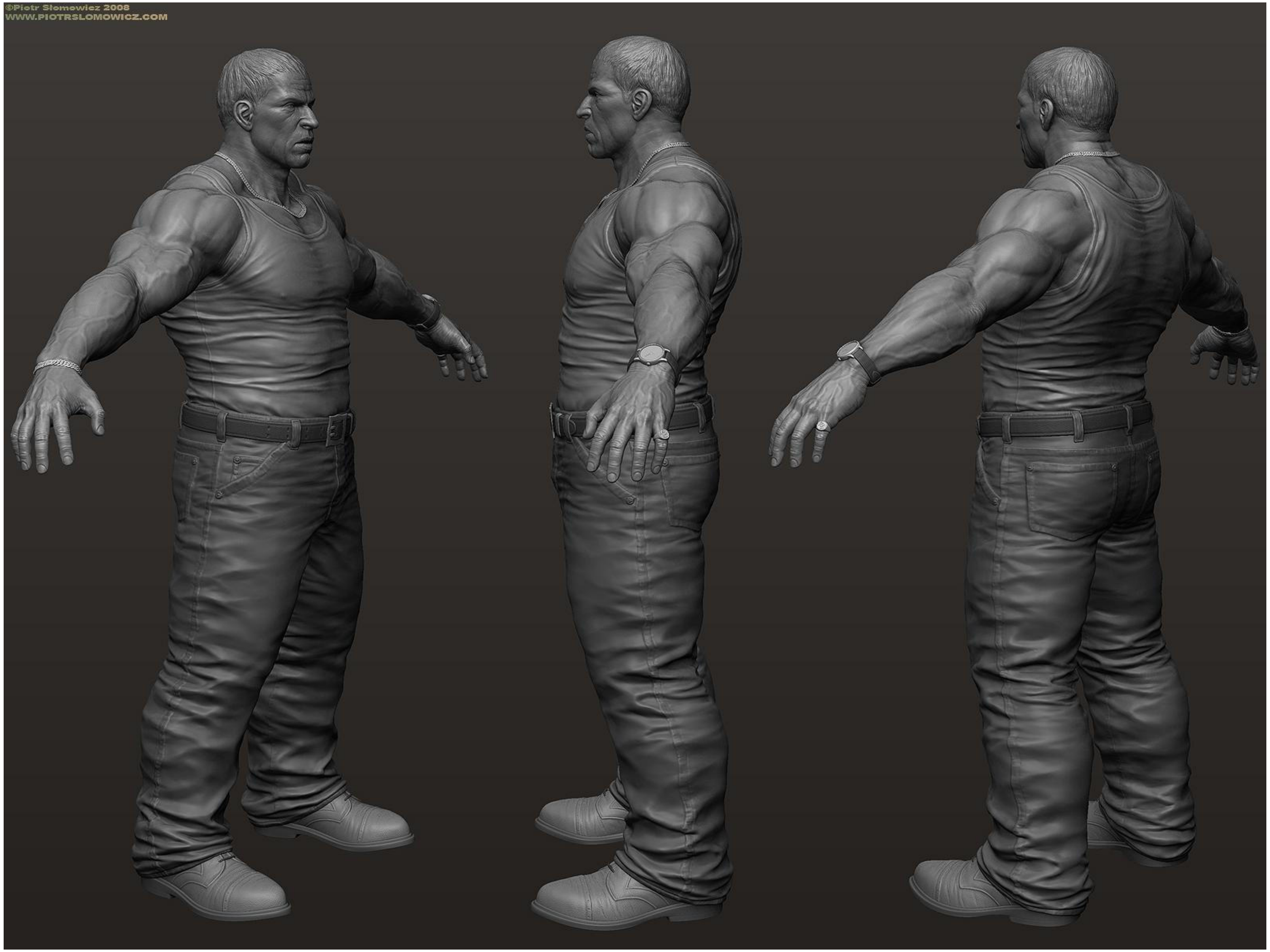
Normal Map Example

+



Model by Piotr Slomowicz

Normal Map Example



Normal Map Example

Normal Mapping + Environment Mapping

- Normal and parallax mapping combines beautifully with environment mapping

EMNM (World Space)

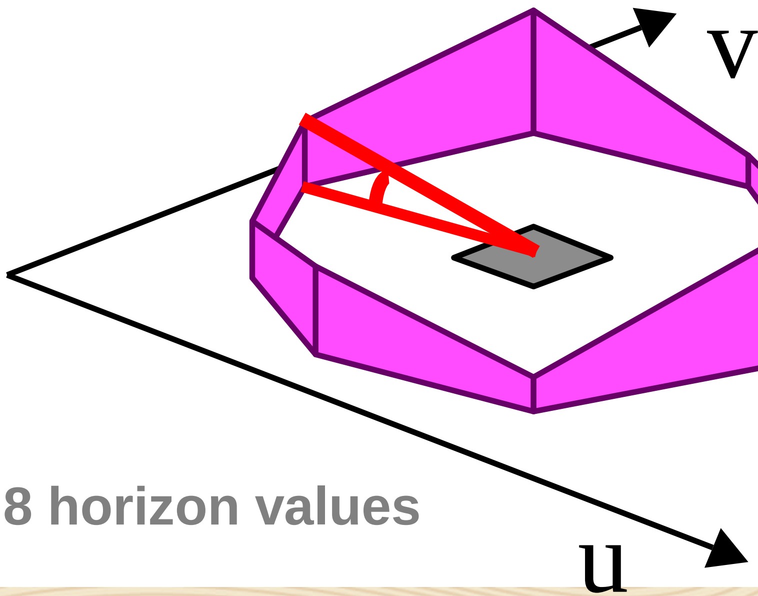
- For each vertex
  - Transform V to world space
  - Compute tangent space to world space transform (T, B, N)
- For each fragment
  - Interpolate and renormalize V
  - Interpolate frame (T, B, N)
  - Lookup N' = texture(s, t)
  - Transform N' from tangent space to world space
  - Compute reflection vector R (in world space) using N'
  - Lookup C = cubemap(R)

Normal and Parallax Normal Map Issues

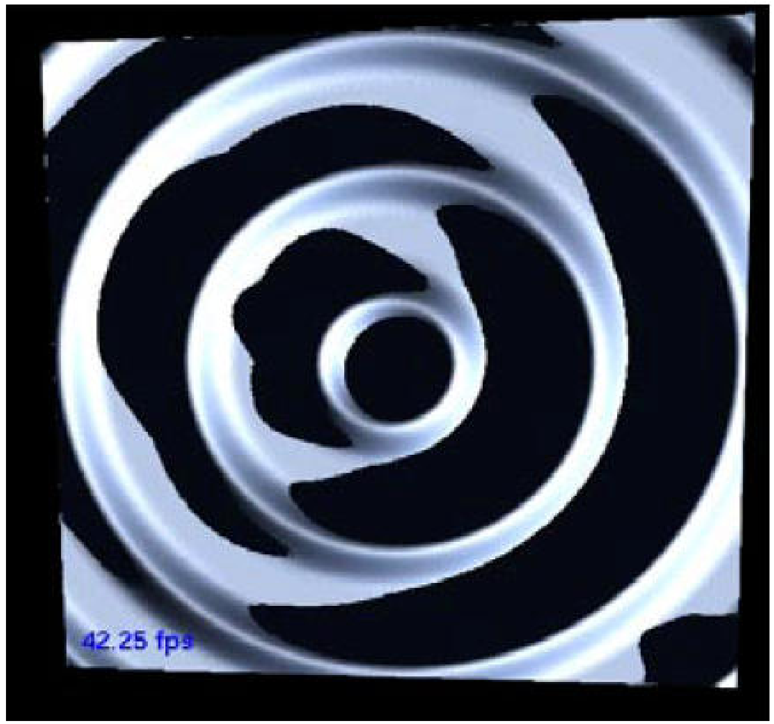
- Artifacts
  - No inter-shadowing
  - Silhouettes still edgy
  - No parallax for normal mapping
- Parallax normal mapping
  - No occlusion, just distortion
  - Not accurate for high frequency height fields
    (local constant height field assumption does not work)
  - No silhouettes

Horizon Mapping

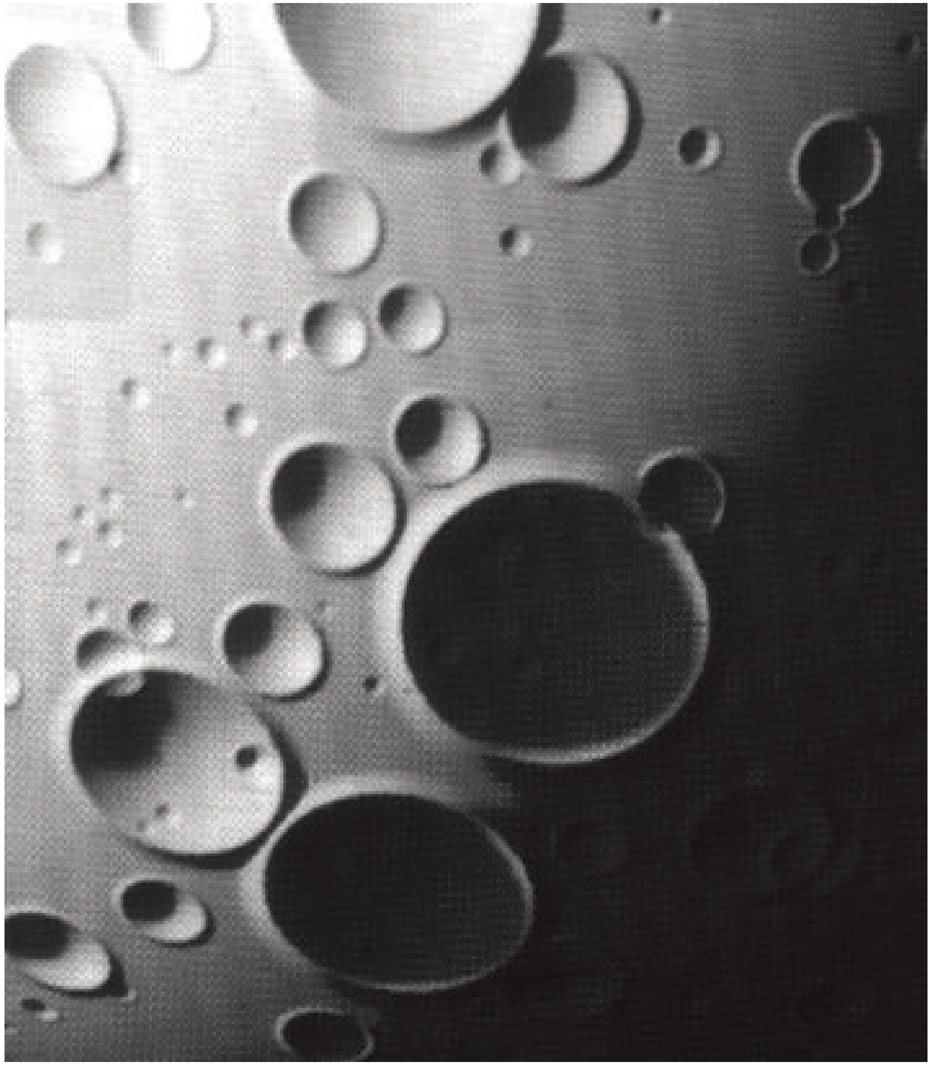
- Improve normal mapping with (local) shadows
- Preprocess: compute *n* horizon values per texel
- Runtime:
  - Interpolate horizon values
  - Shadow accordingly

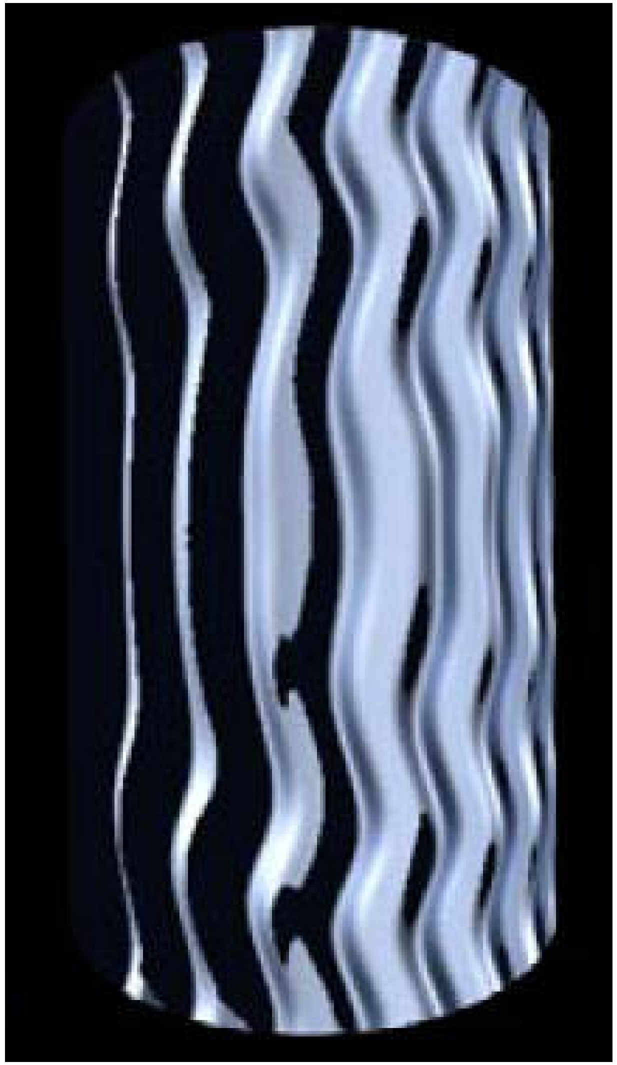


8 horizon values



Horizon Mapping Examples





Relief Mapping

Relief Texture Mapping

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

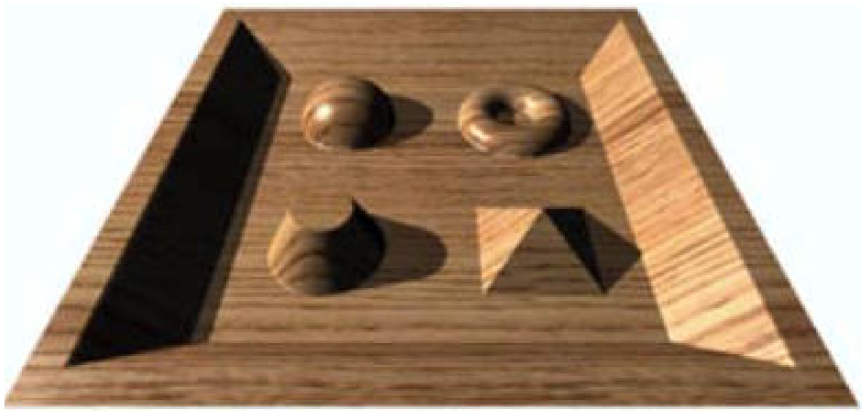
Relief Texture Mapping

- 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

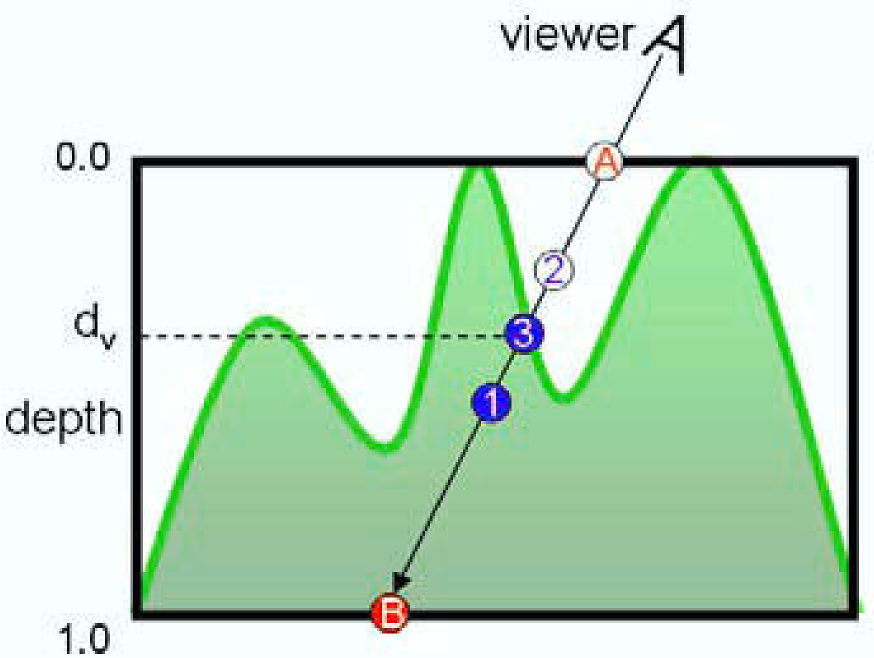
Parallax mapping

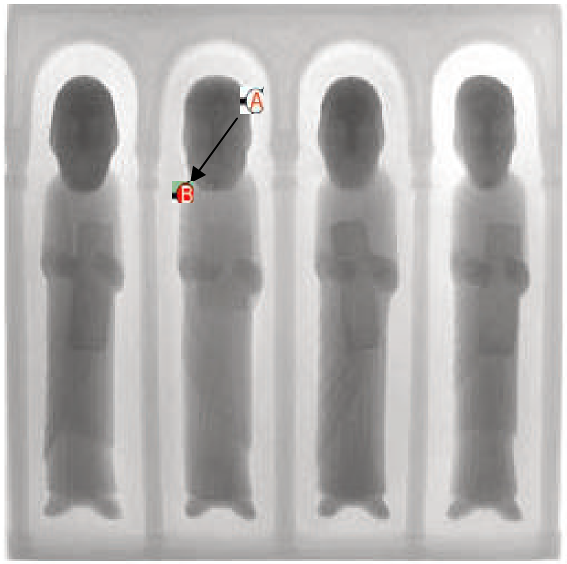


Relief mapping

Mapping Relief Data

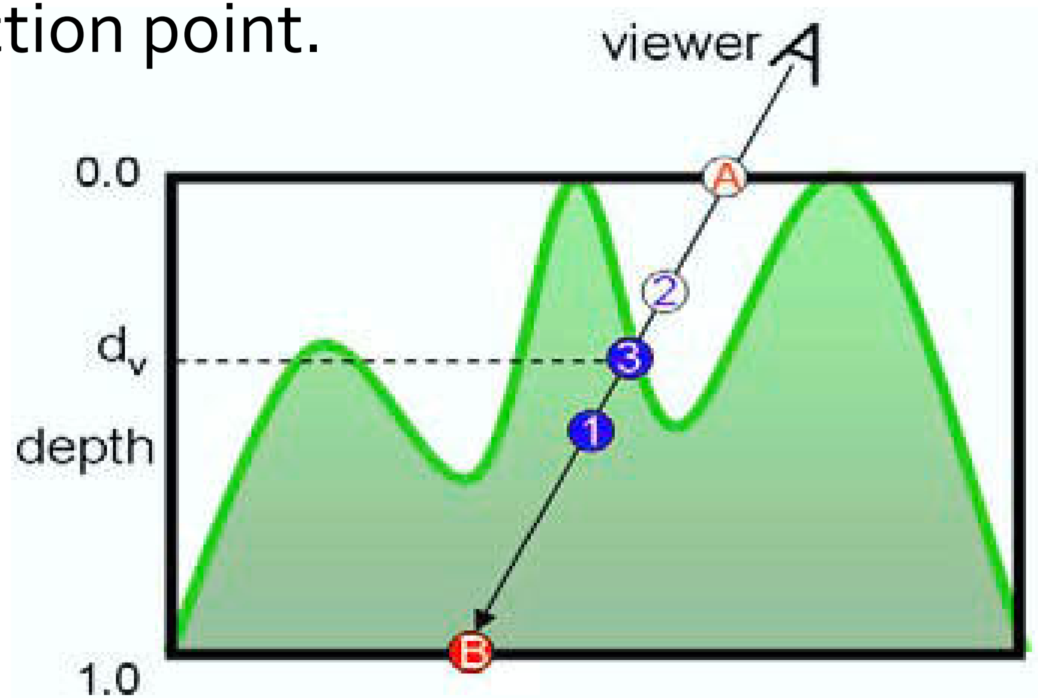
- Compute viewing direction, VD
- Transform VD to tangent space of fragment
- Use VD’ and texture coords (s,t) to compute the texture coords where VD’ hits depth of 1





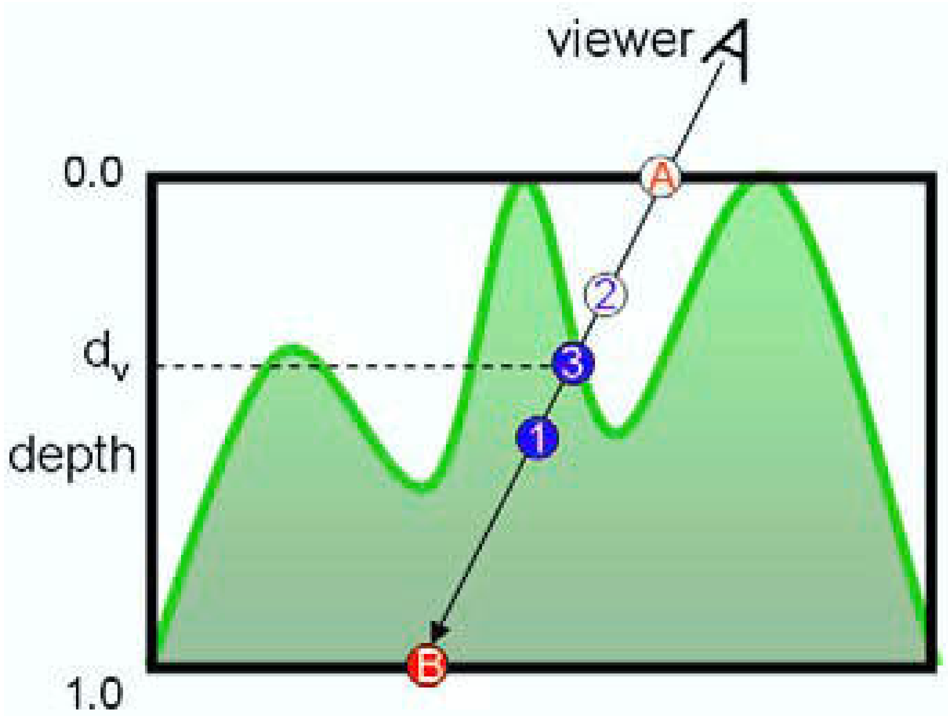
Mapping Relief Data

- 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.



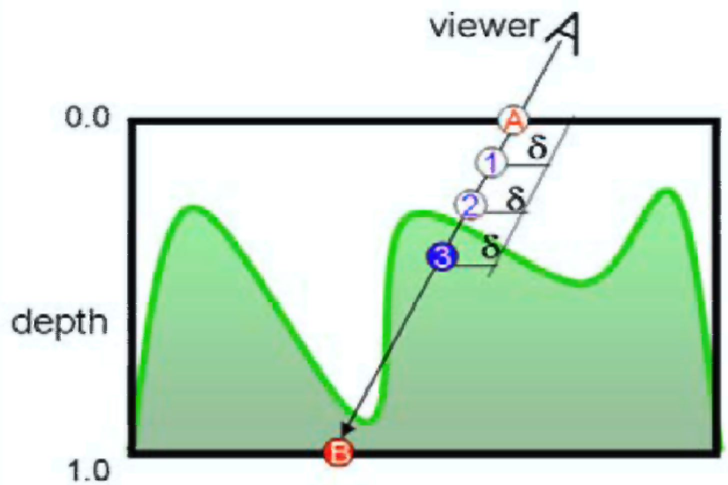
Binary Search

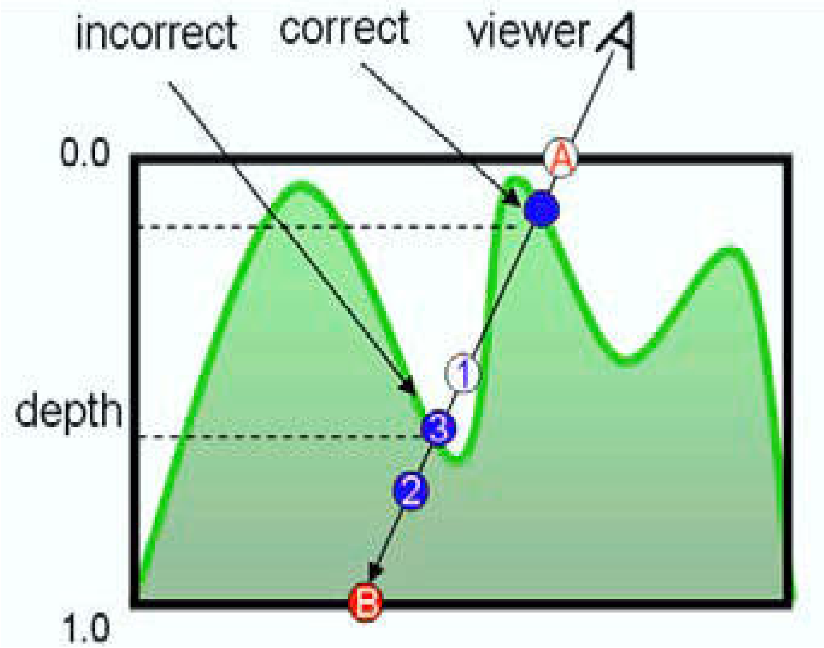
- Start with A-B line
- At each step:
  - Compute middle of interval
  - Assign averaged endpoint texture coordinates and depth
  - Use averaged tex coords to access depth map
  - If stored depth value is less than computed depth value, the point is inside the surface
  - Proceed with one endpoint in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



Combined Search

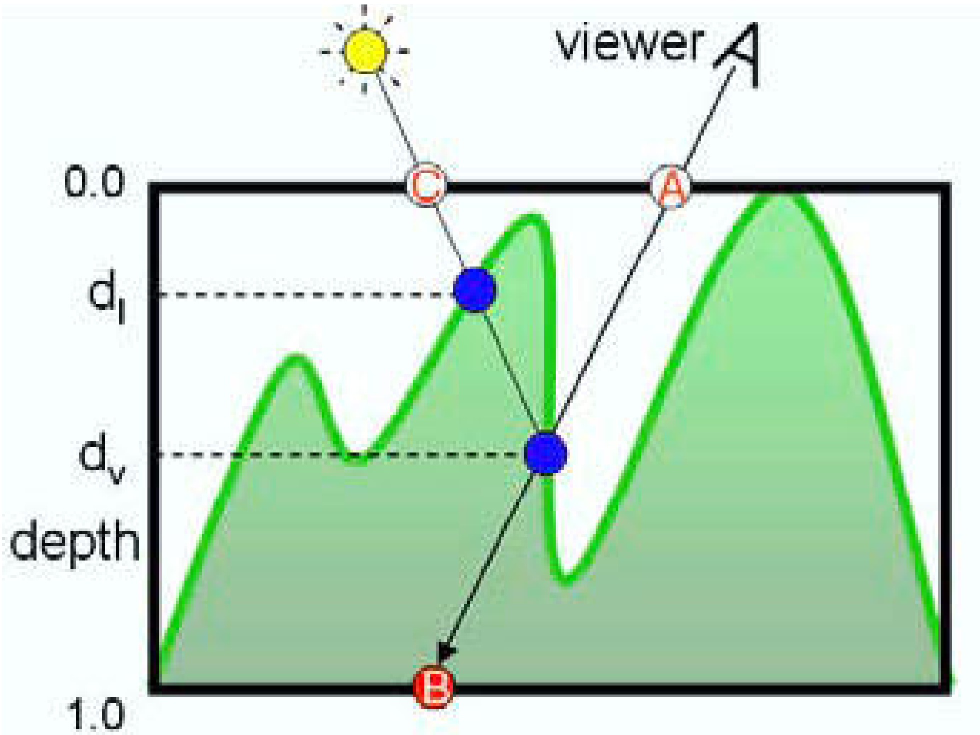
- To find first point under surface, start at A, advance ray by δ
- δ is a function of the angle between VD' and interpolated fragment normal
- Proceed with binary search (with less iterations)





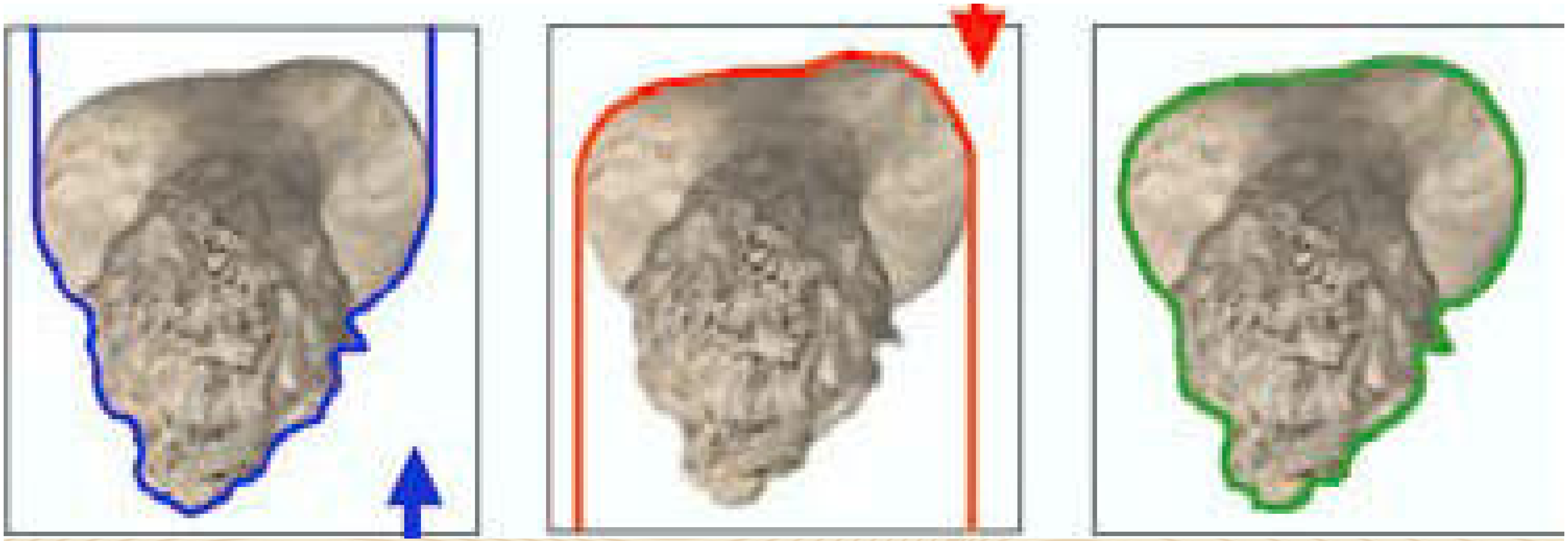
Shadowing

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

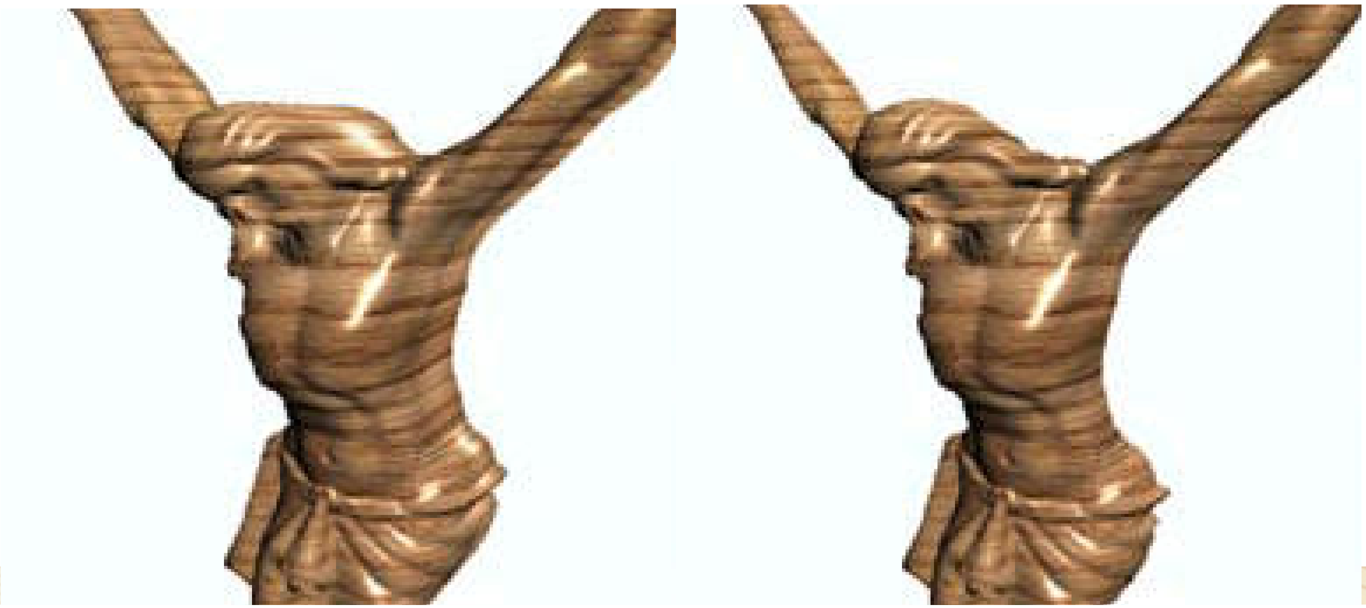


Dual Depth Relief Textures

- 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

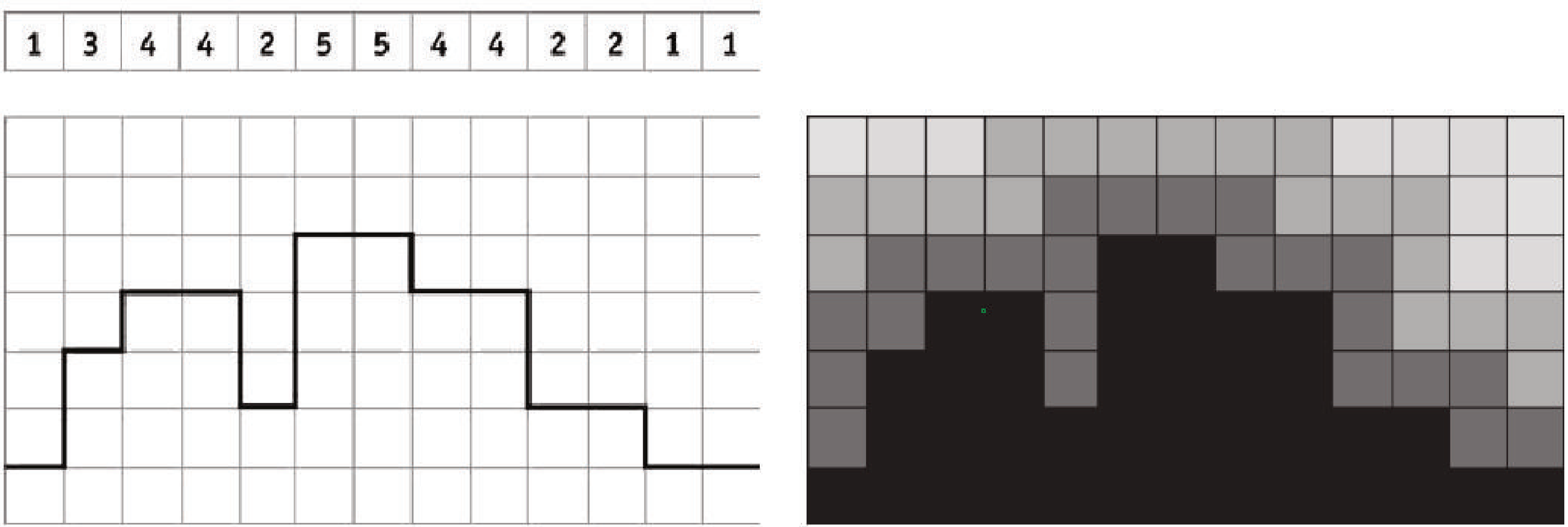


Dual Depth Relief Textures



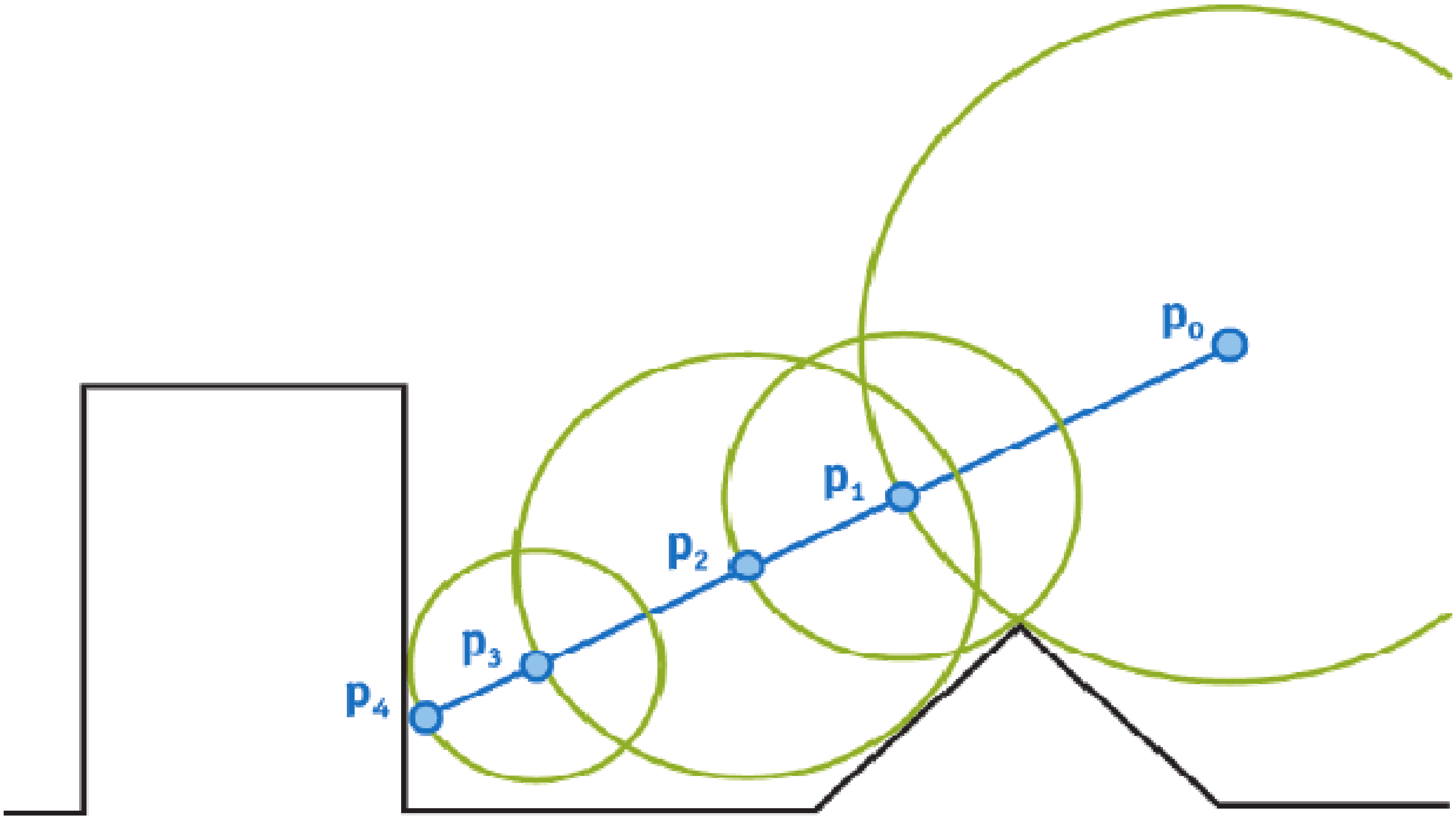
Sphere Tracing

- Store distance to closest surface in 3D map



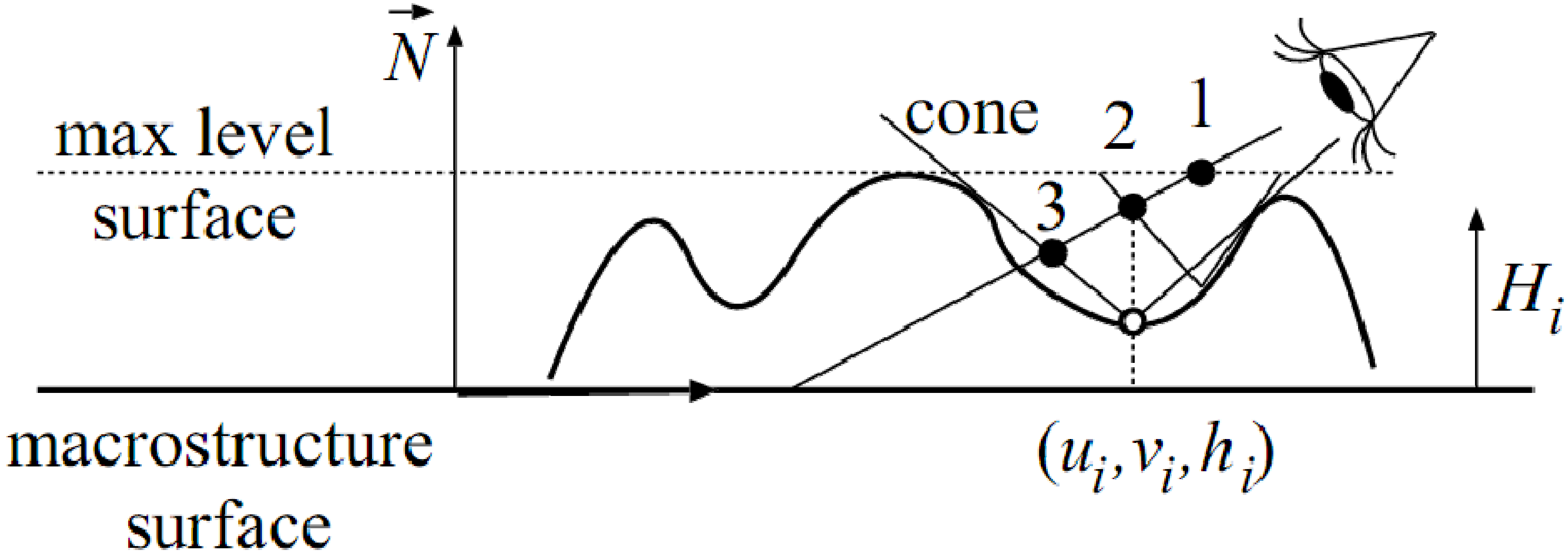
Sphere Tracing

- Distance is sphere radius for search step

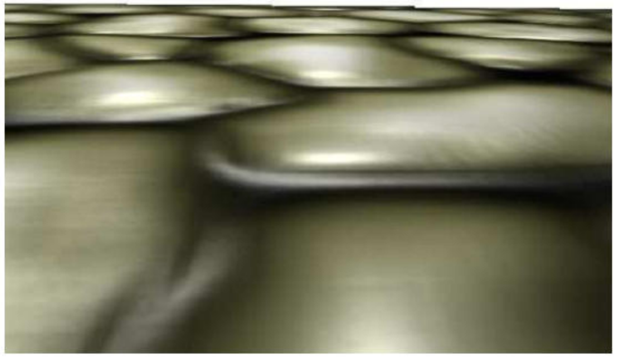


Cone Stepping

- Texel stores cone of empty space above
- Only store opening angle

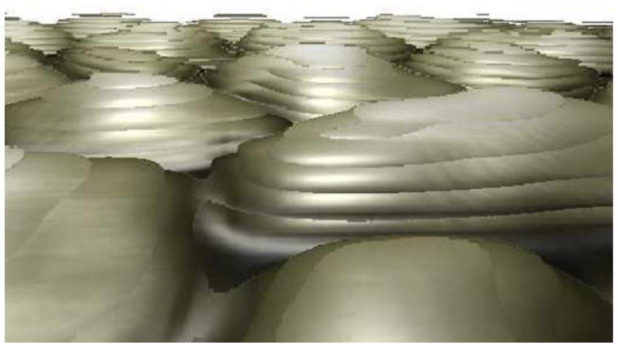


texture mapping

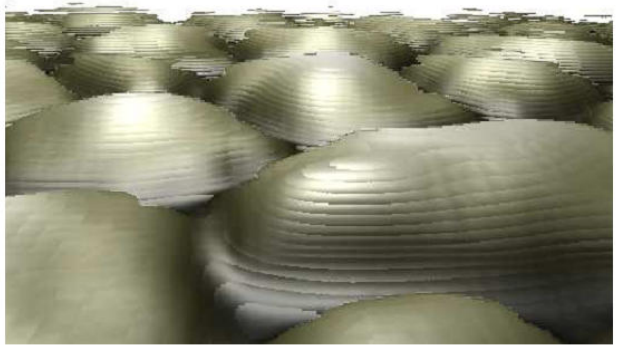


bump mapping

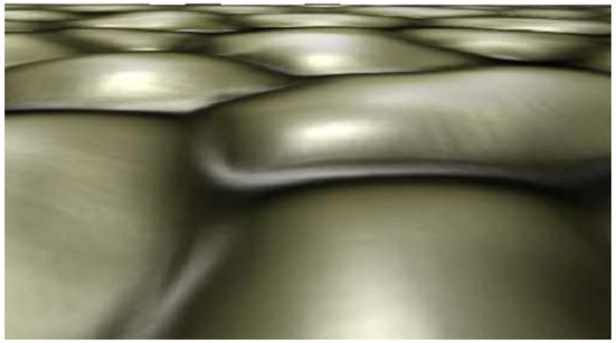
parallax mapping



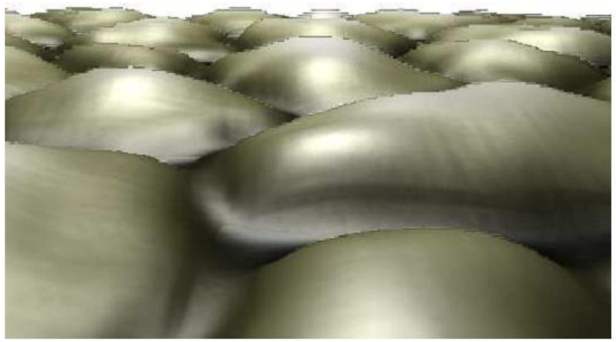
ray marching



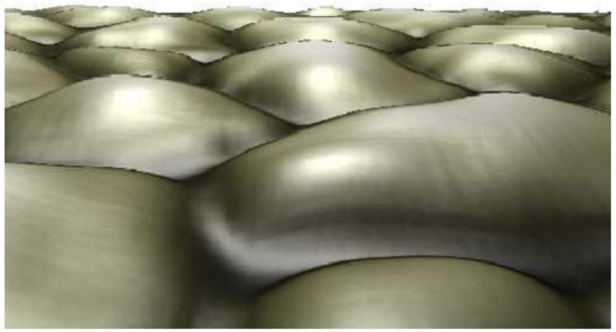
binary search



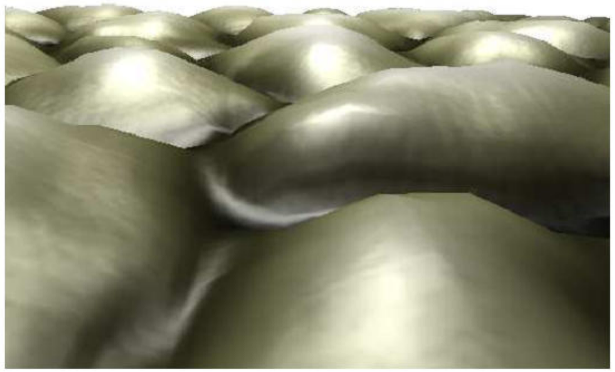
sphere tracing



relief mapping



cone stepping

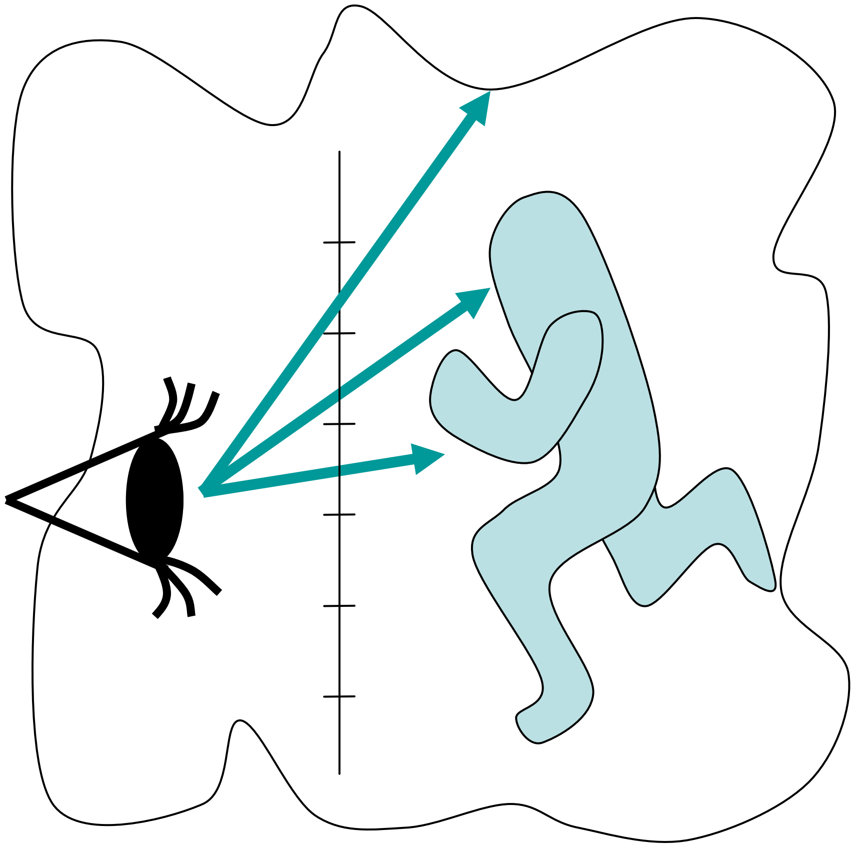


per-vertex displacement mapping

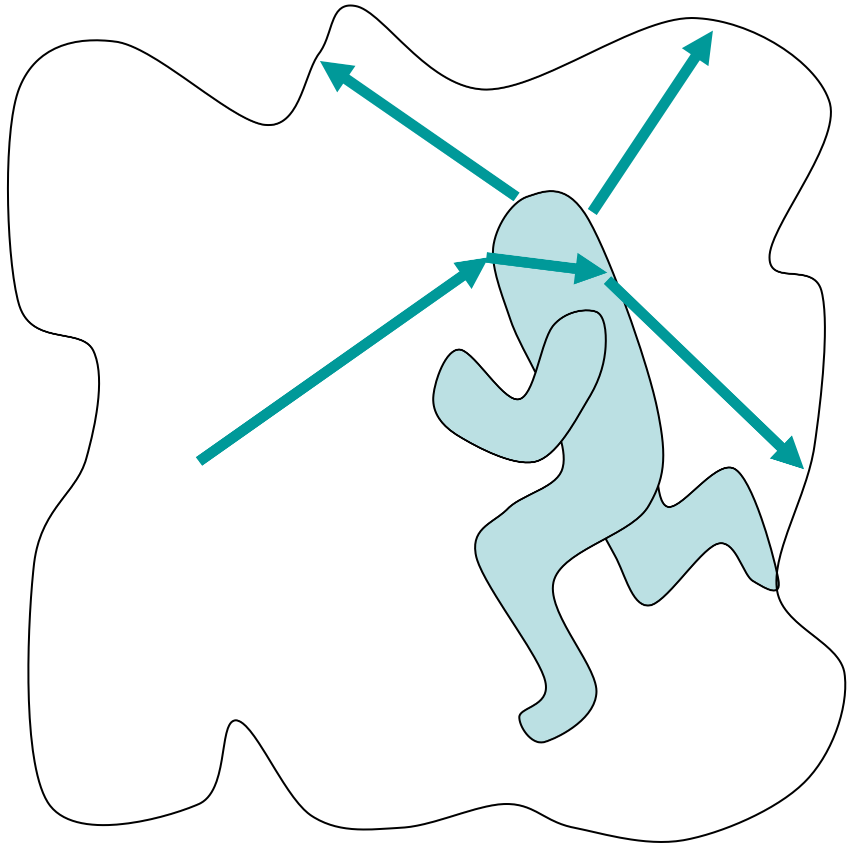
Speed considerations

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

Rasterization versus Ray-Tracing

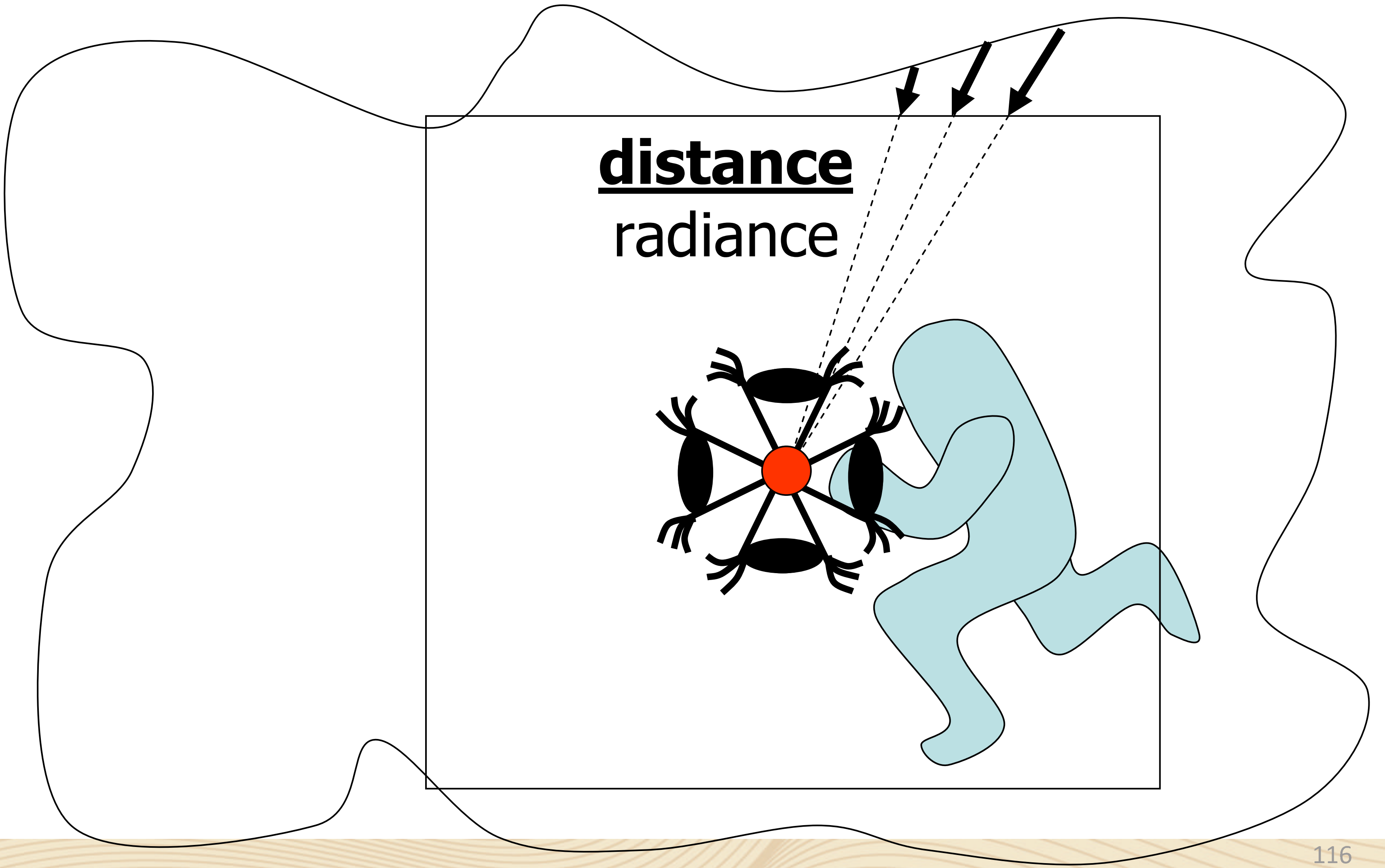


Inremental rendering on the GPU

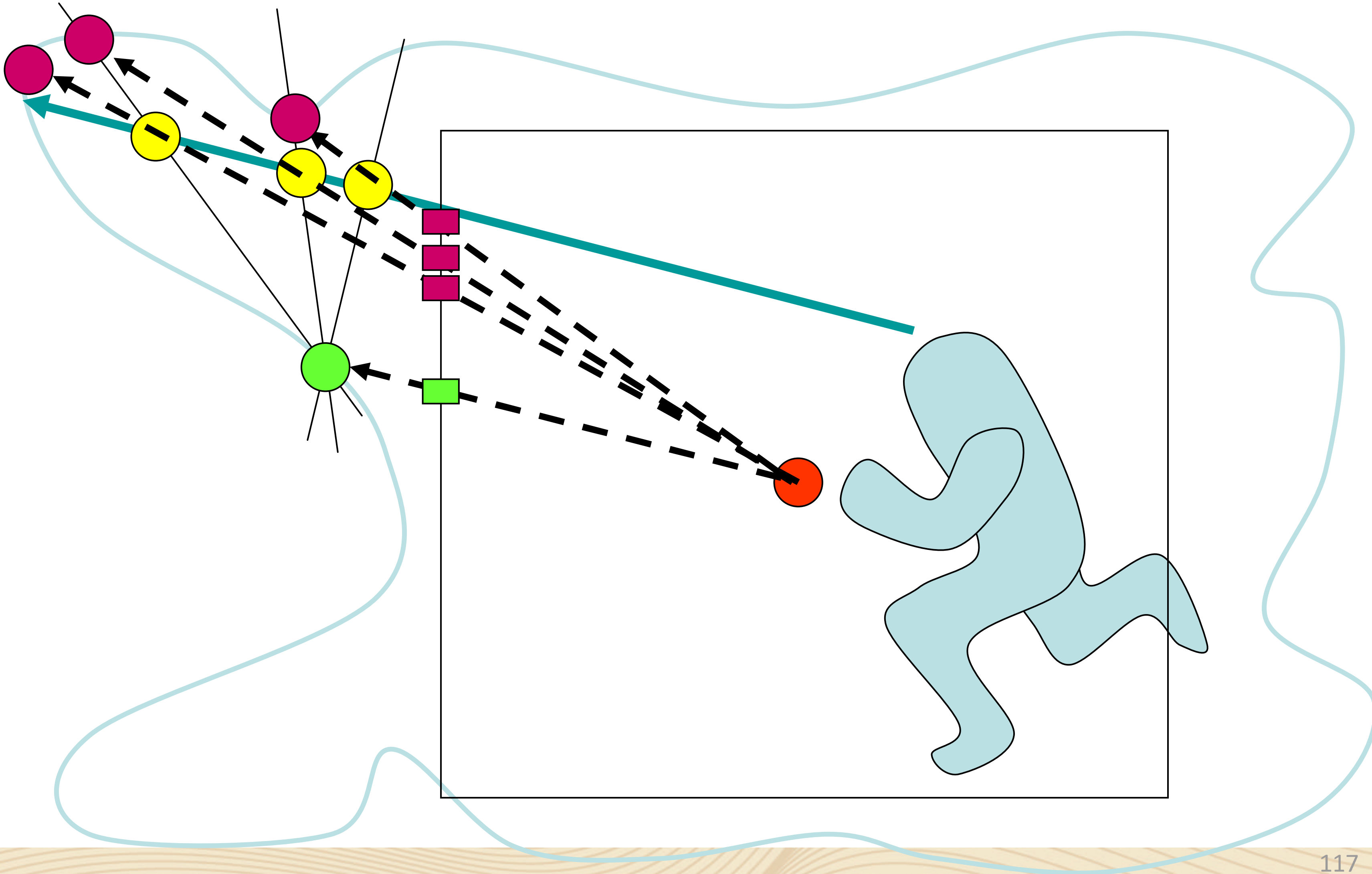


Non-coherent Ray tracing

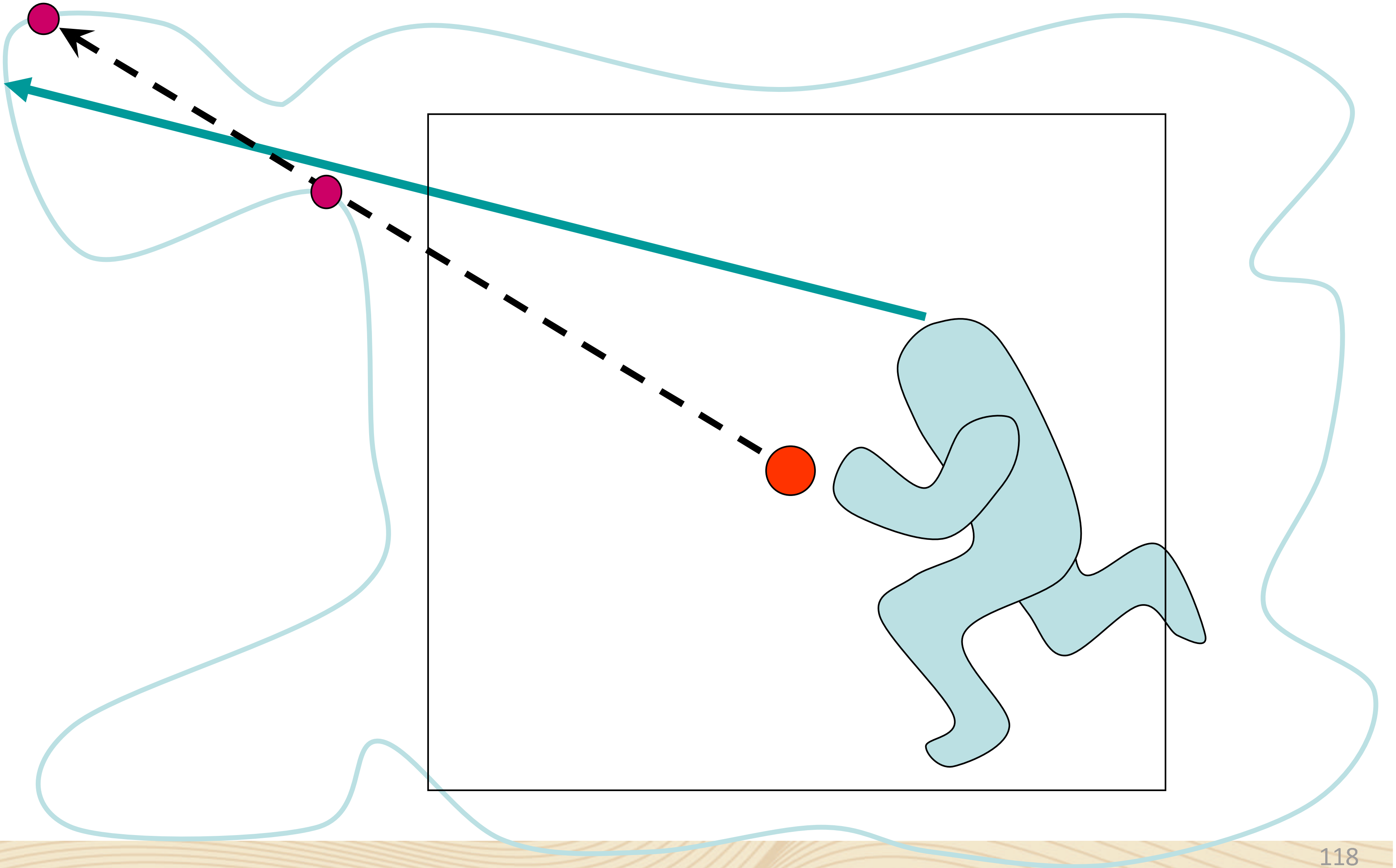
Distance Impostors



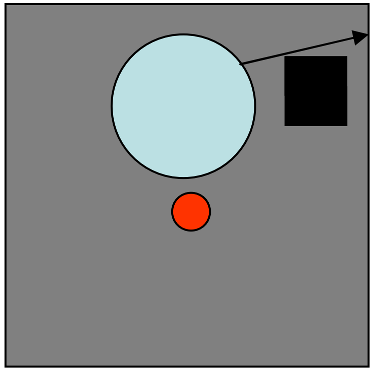
Ray-Tracing with Distance Impostors

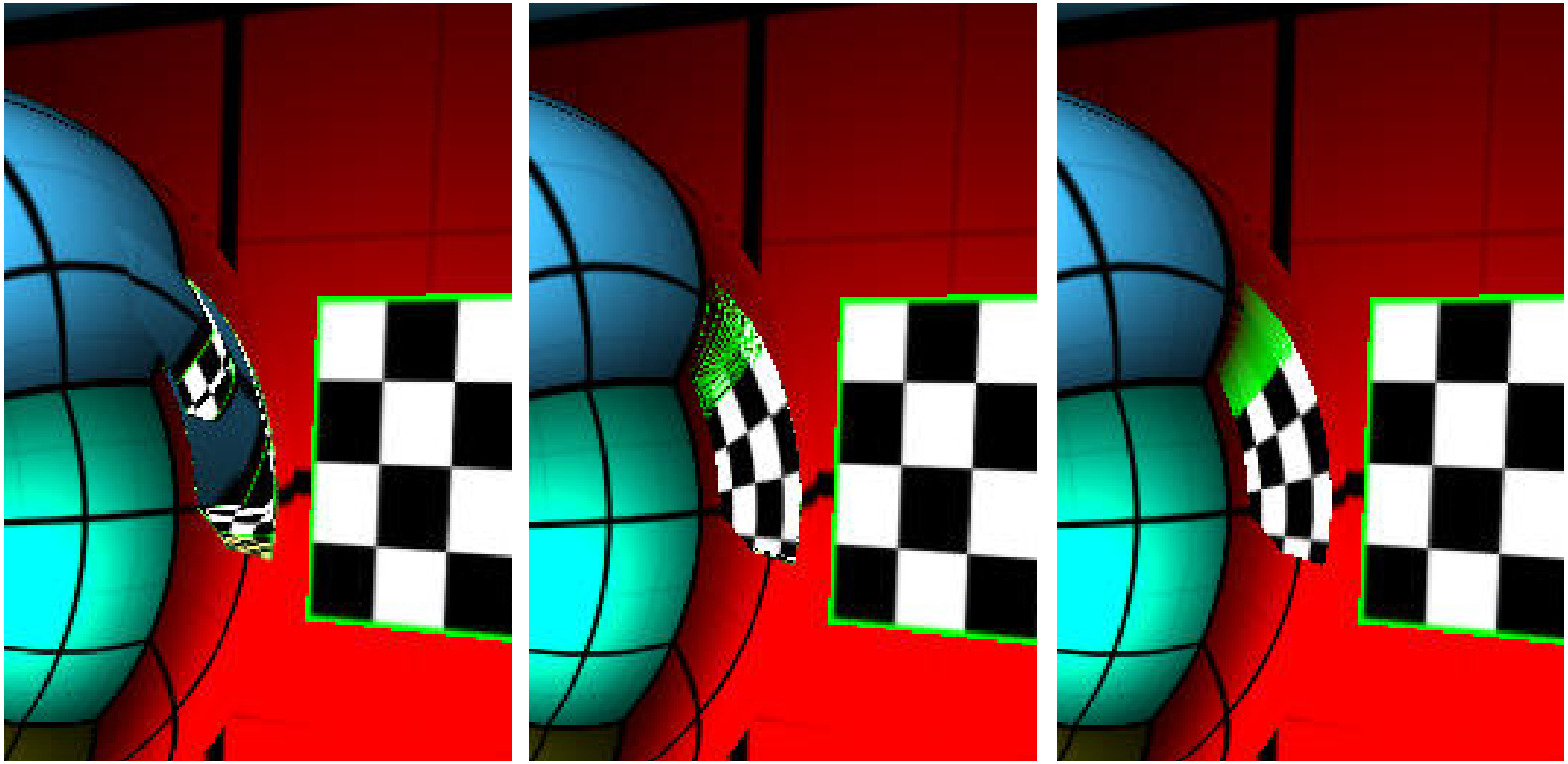


Approximation



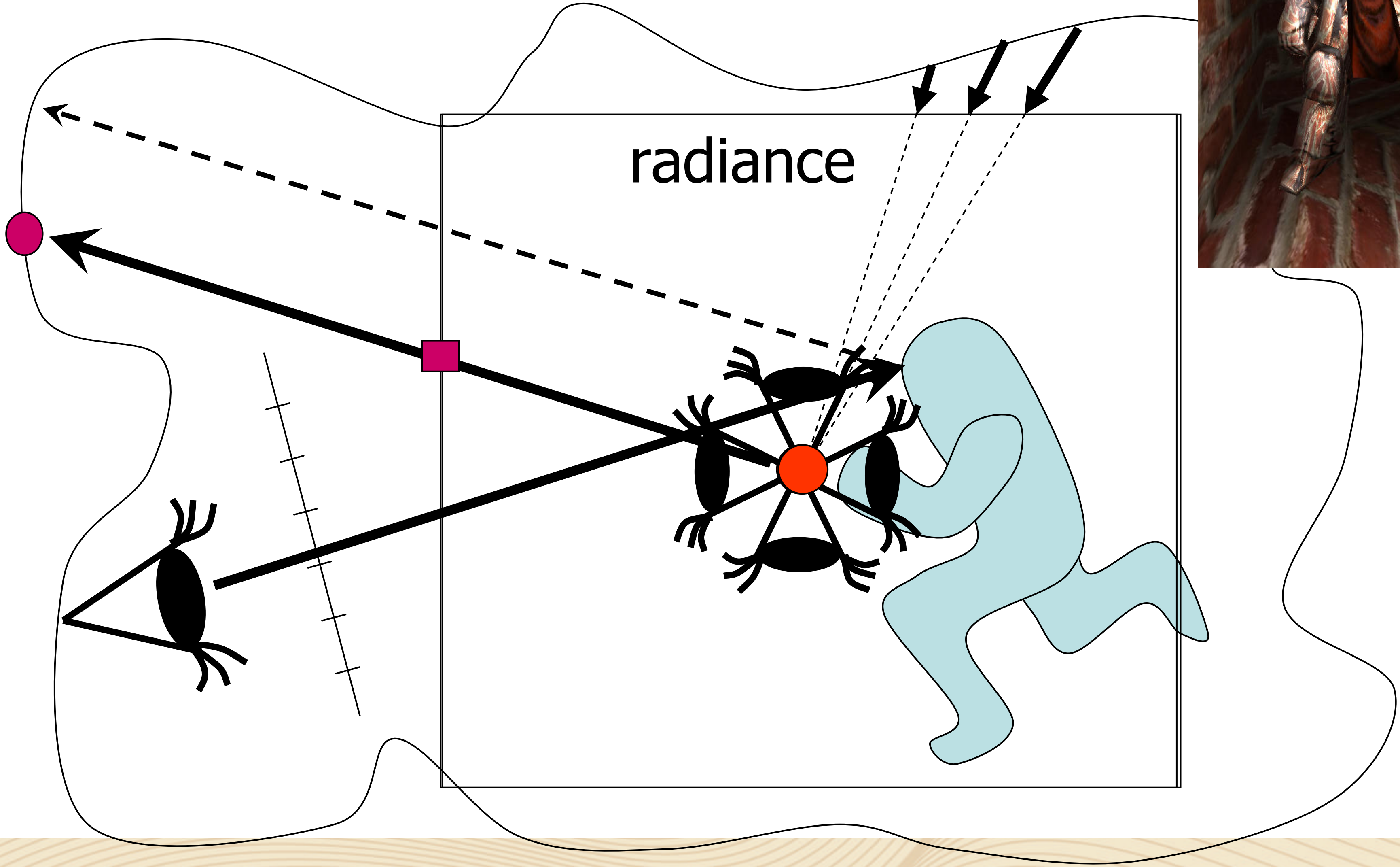
Approximation Error

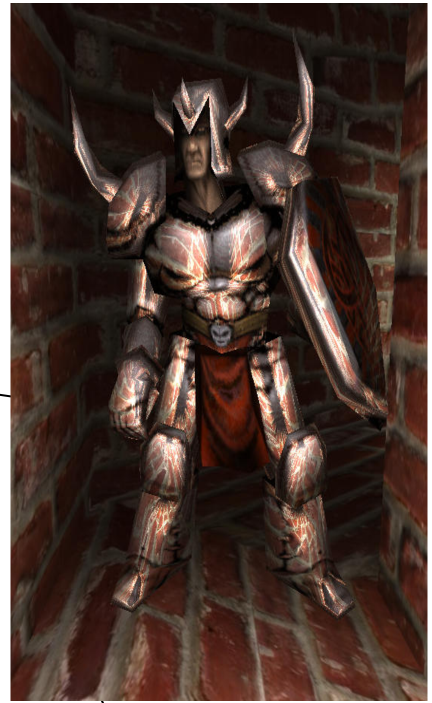




1 iteration | 4 iterations | 8 iterations

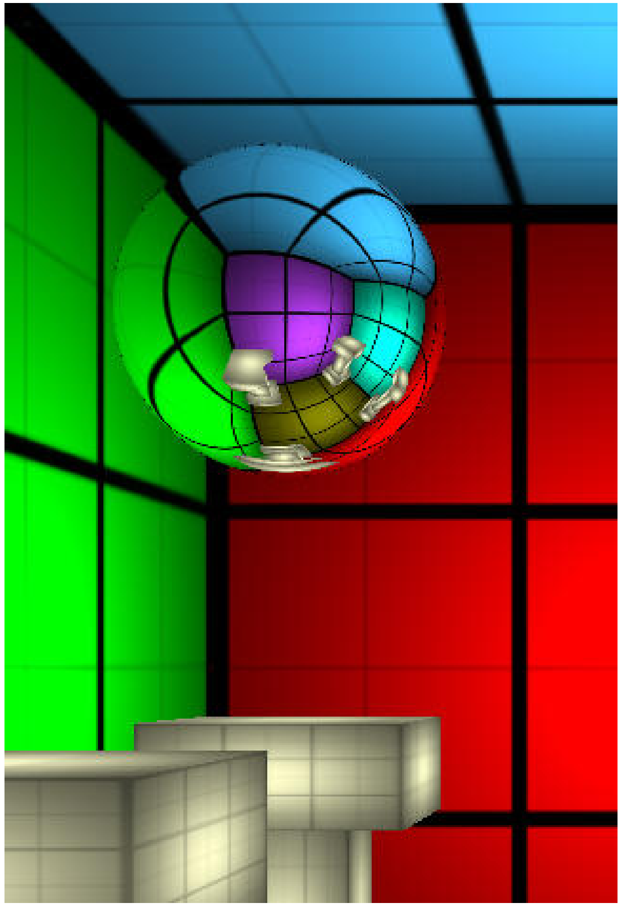
Reflections





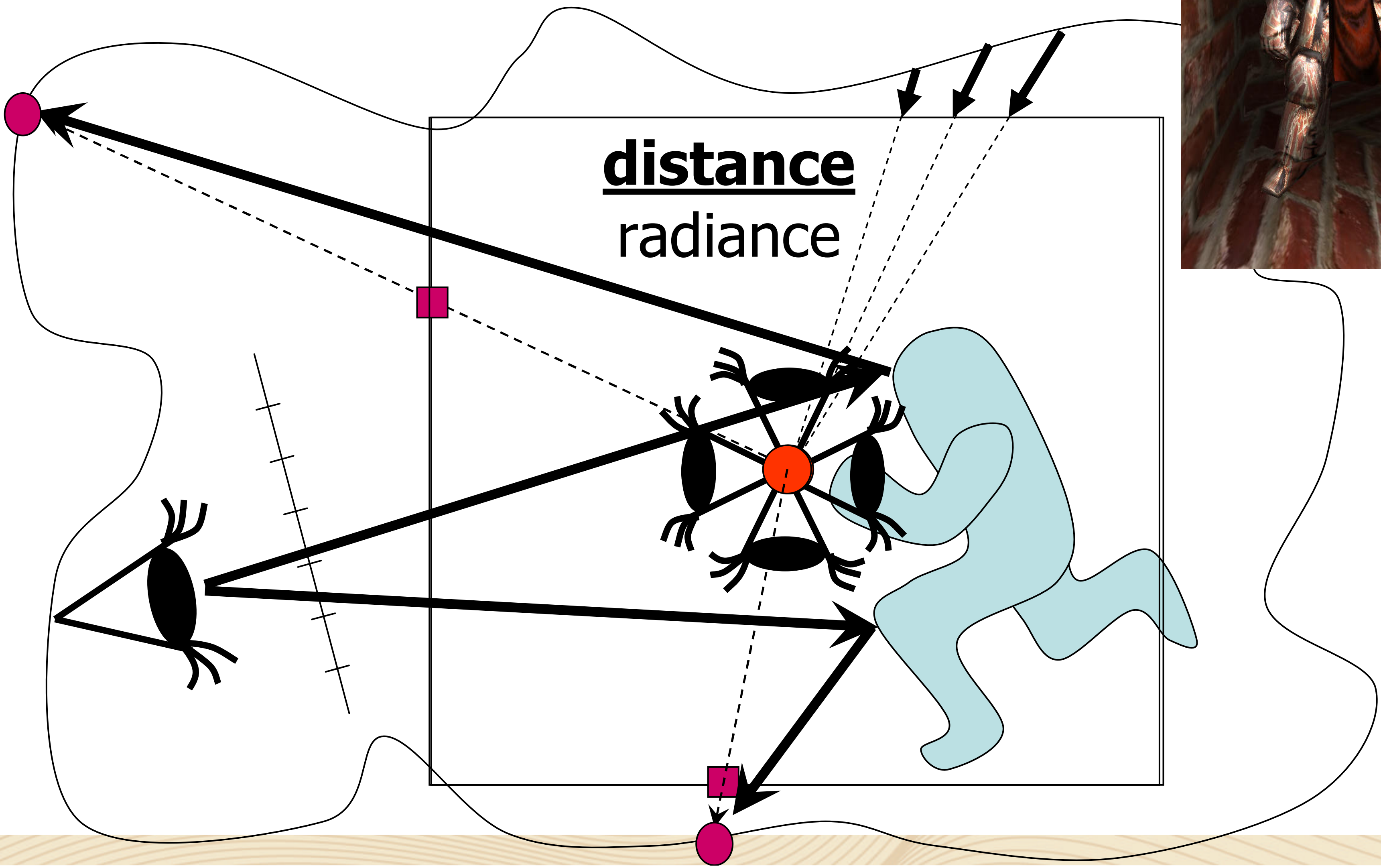
Problems of environment map based reflections

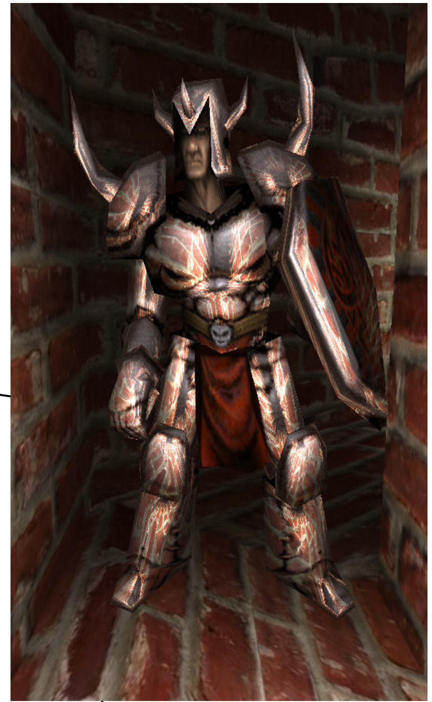
Environment map

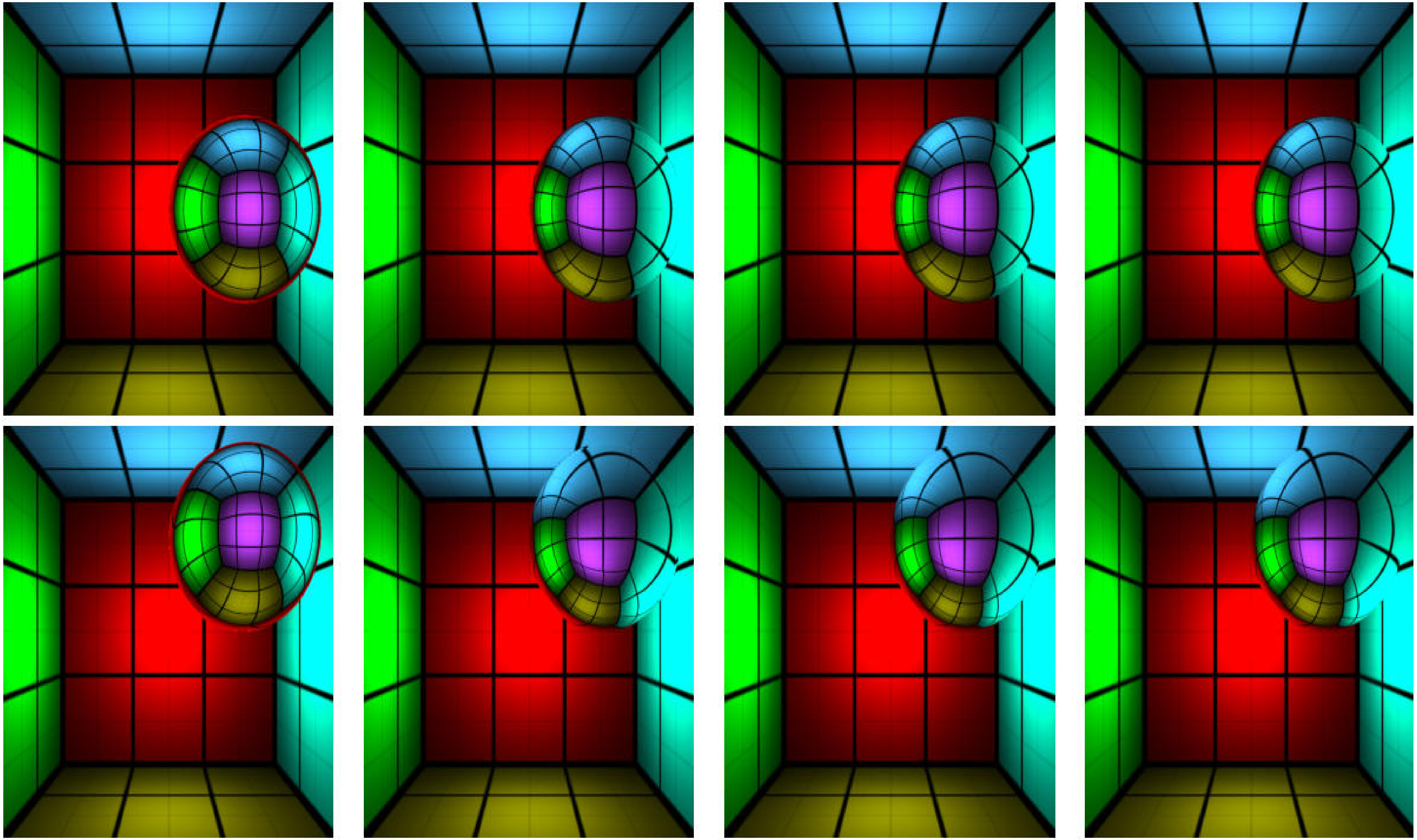


Reference

Localized Reflections







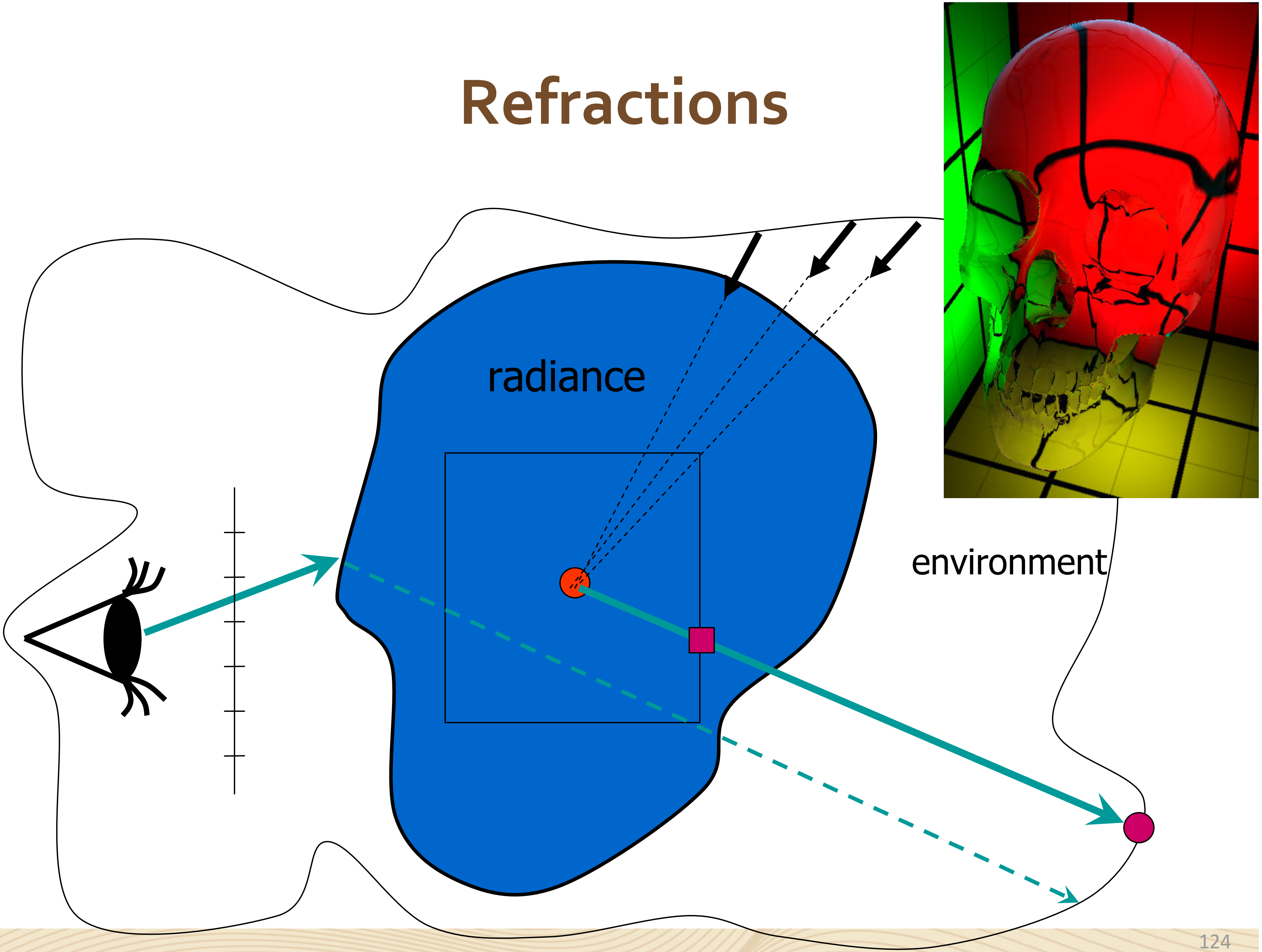
classical environment map
642 FPS

distance impostor
447 FPS

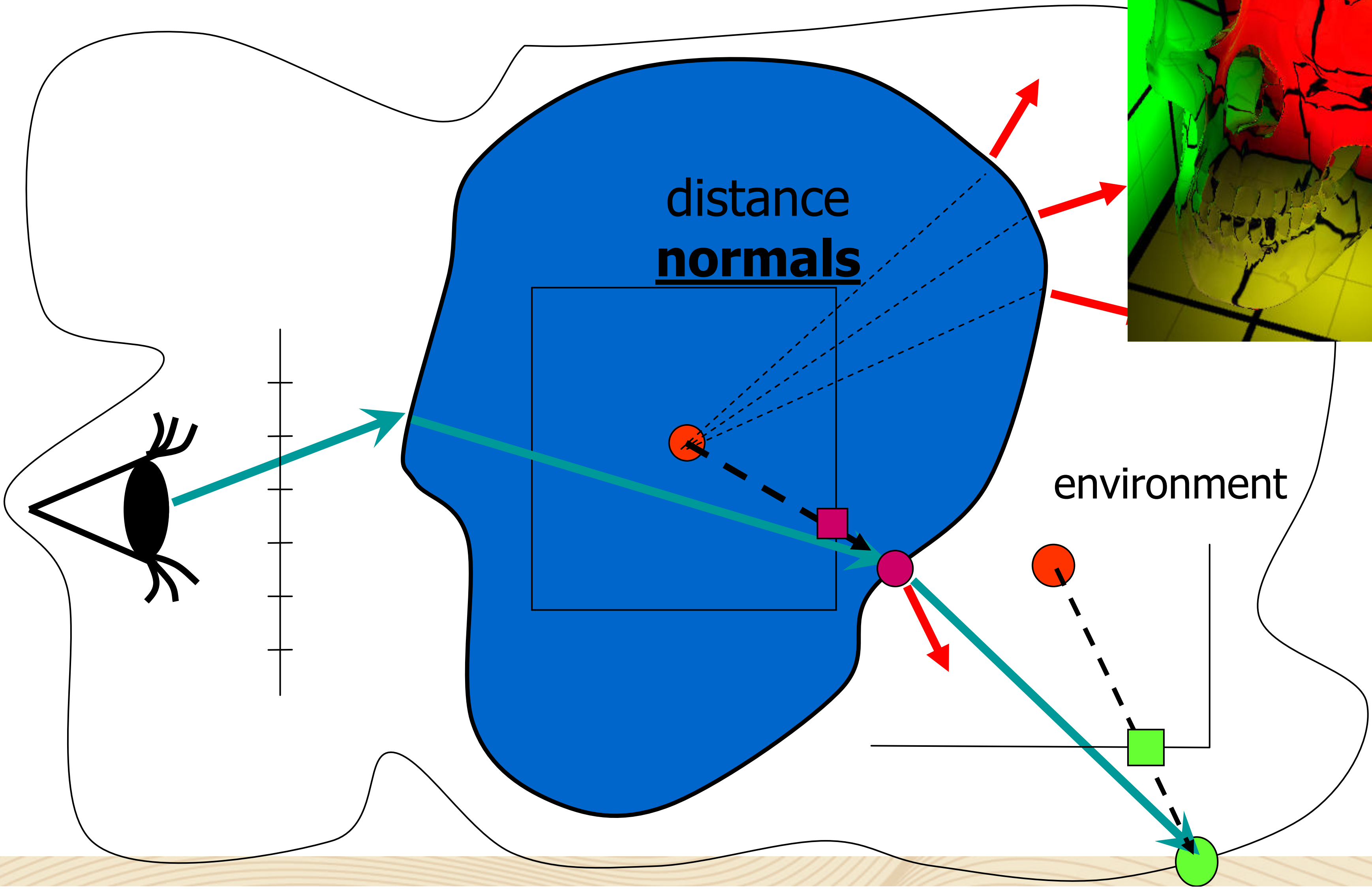
+1 iteration
323 FPS

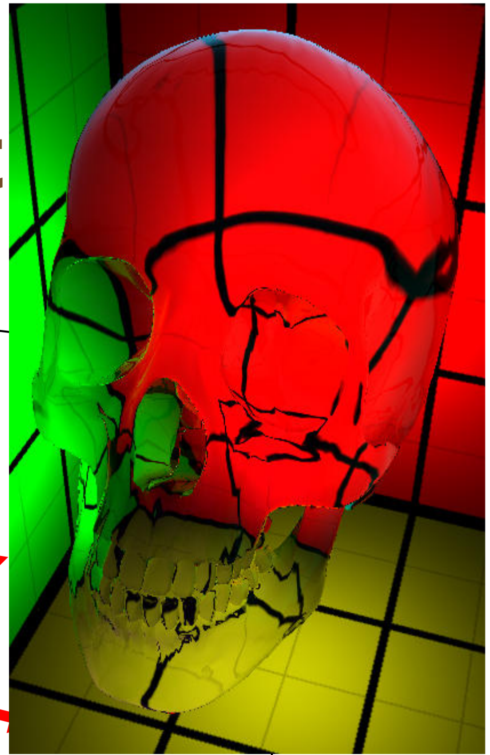
ray traced reference

Refractions

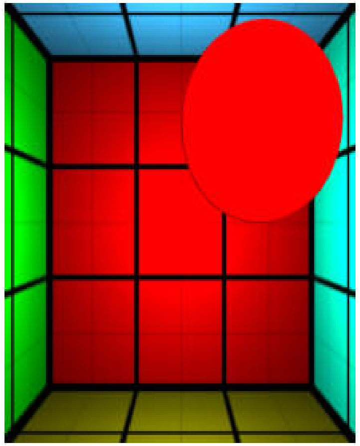


Multiple Localized Refract

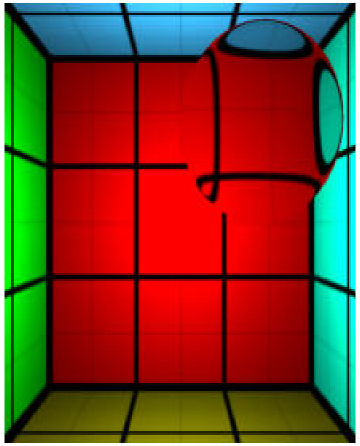




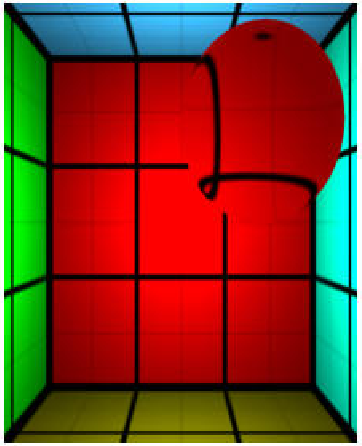
Multiple Localized Refractions



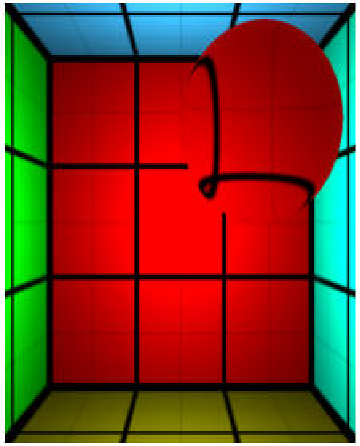
classical
environment map
804 FPS



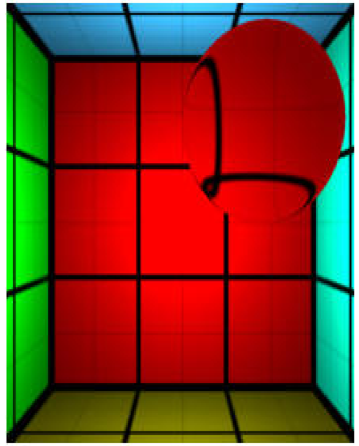
distance impostor
single refraction
695 FPS



distance impostor
double refraction
508 FPS



+1 iteration
double refraction
249 FPS



ray traced
reference