Depth Image Approximation of Geometry & Applications

Voicu Popescu

Geometry approximation

- Definition
  - Alternative representation of geometry
  - Smaller cost but comparable effect
  - “Impostor”—looks like but is not the “real thing”
- Motivation
  - Acceleration of expensive rendering effects
  - Replace geometry with approximation for faster frame rates and same / similar quality

Example: specular reflections

- Projection followed by rasterization cannot render reflect*ed* triangles
  - No closed form projection
  - Non-linear rasterization
- Per-pixel reflected ray is easy to compute
  - Conventional rasterization of reflect*or* triangle
  - Normal interpolation

Example: specular reflections

- Projection followed by rasterization cannot render reflect*ed* triangles
- Per-pixel reflected ray is easy to compute
- Intersecting per-pixel reflected ray with scene geometry is challenging

Example: specular reflections

- Per-pixel reflected ray is easy to compute
- Intersecting per-pixel reflected ray with scene geometry is challenging
- Idea: approximate scene geometry to simplify intersection with reflected ray

Other examples

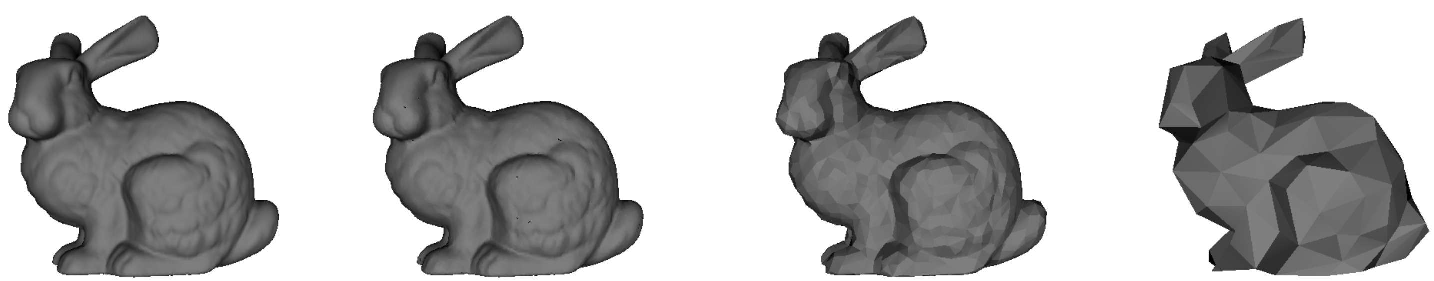
- Reducing triangle load in conventional rendering
- Refractions
- Hard and soft shadows
- Surface geometric detail

Geometry approximations

- Simplified geometry
- Cube map
- Billboard
- Depth image

Geometry simplification

- Reducing the number of polygons
- Goals: maximize quality over cost
  - maximize similarity to original geometry (given a specific metric)
  - minimize number of polygons



69,451 tris | 30,994 tris 1% error | 2,502 tris 5% error | 251 tris 15% error

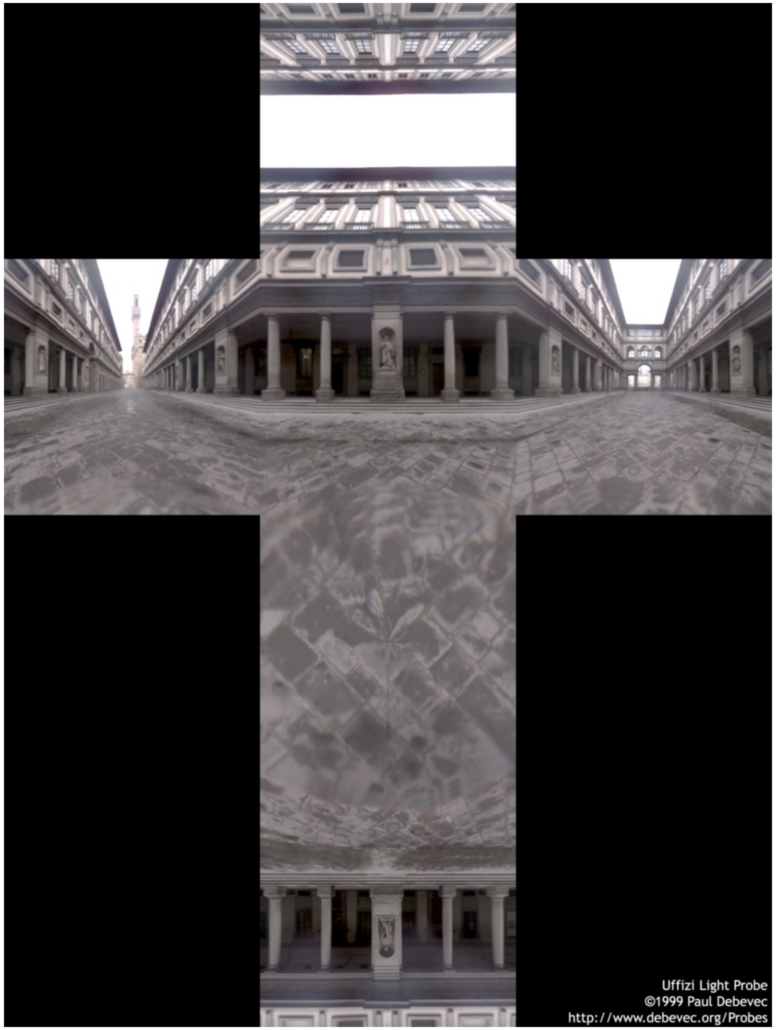
Geometry simplification

- Reducing the number of polygons
- Goals: maximize quality over cost
- Challenges
  - Difficult to do for large meshes with complex topology
  - Difficult to transition smoothly between consecutive levels of detail

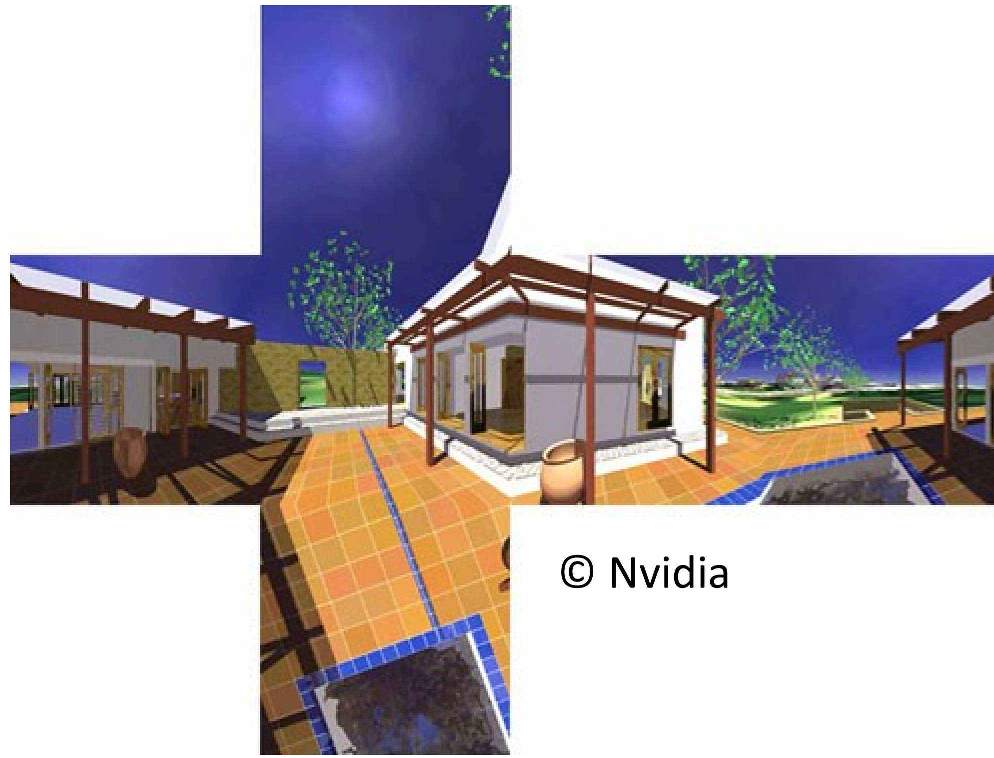
Cubemap

- A panoramic image
  - Other panoramic images are possible (e.g. cylindrical, spherical, etc.)
- Samples in all directions from given point
- Equivalent to 6 image acquired with 6 cameras
  - Same viewpoint, i.e. center of a cube
  - 90^o x 90^o field of view
  - Image frames defined by faces of cube

Cubemap examples



© Debevec



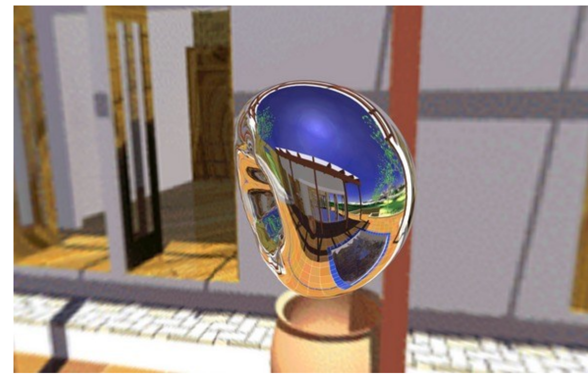
© Nvidia

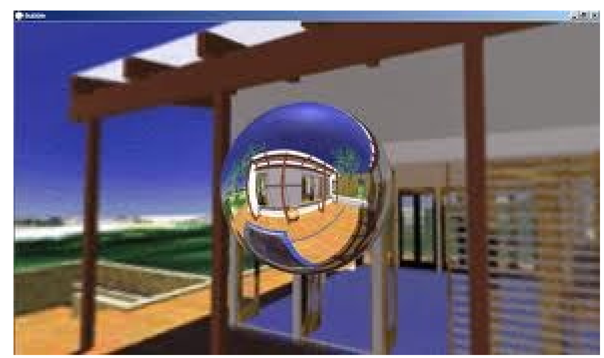
Cubemap

- Advantages
  - Simple to construct
    - Synthetic scenes: rendering 6 images
    - Real-world scenes: acquisition with multiple cameras, or with combination of camera(s) and mirror(s)
  - Simple to use: easy to lookup a ray
    - Find the cubemap face intersected by the ray
    - Find intersection point P
    - Lookup color in cubemap face image (texture) at P

Cubemap applications

- Rendering distant geometry
  - Mountains, clouds
- Specular reflections
- Refractions
- Used by most consumer interactive graphics applications, on most platforms
  - Games on PS, Xbox, PCs (GPUs)





© Nvidia

Cubemap

- Disadvantage
  - Drastic approximation of scene geometry
  - Assumes all geometry is infinitely far away

Environment mapping

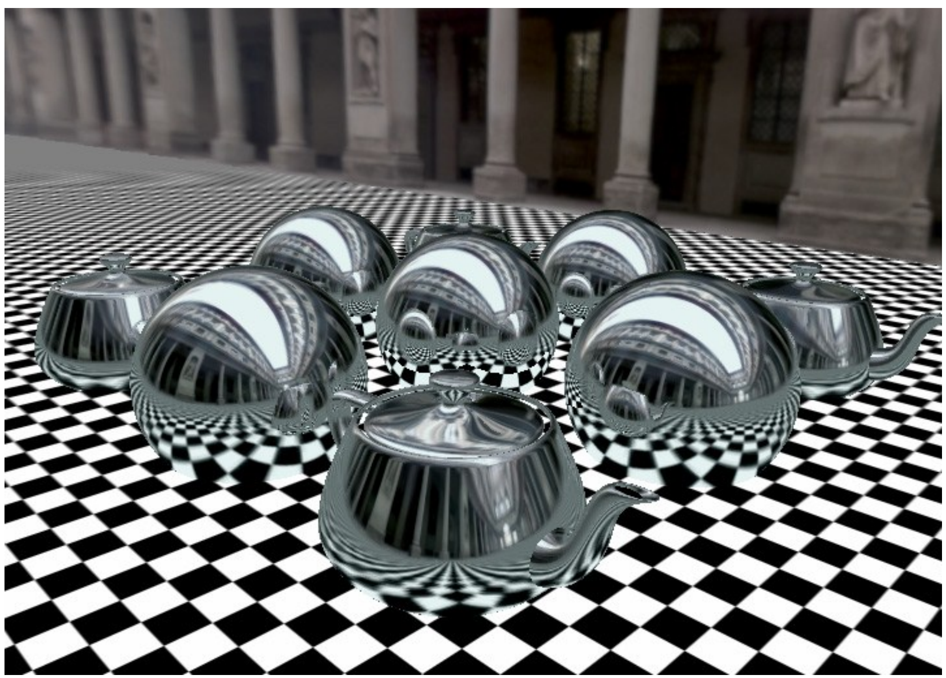
Truth (ray tracing)

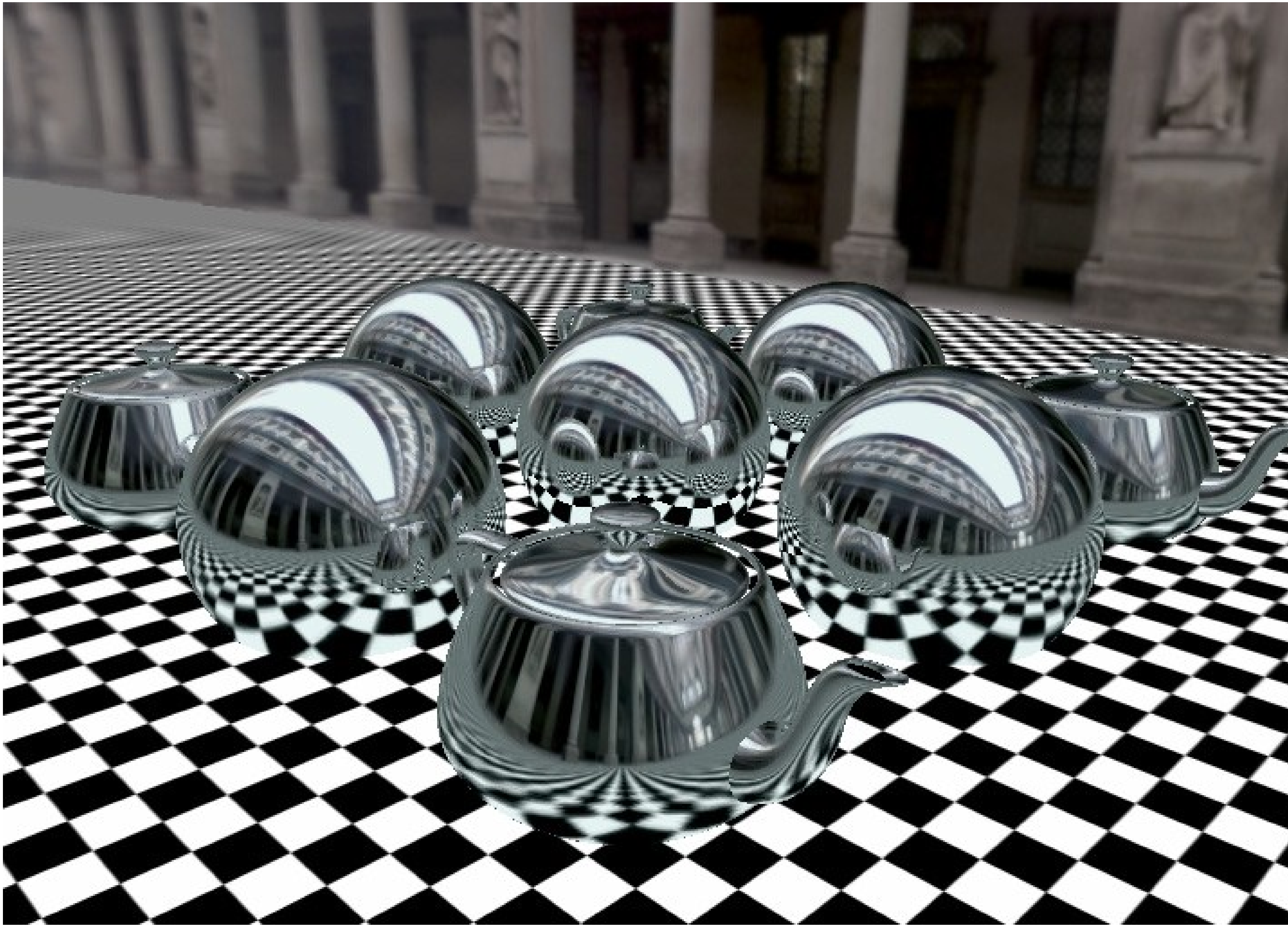
Billboard

- A texture mapped quad
  - Texture shows the geometry replaced (i.e. approximated) by the billboard
  - Texels can be transparent
- Advantages
  - Easy to render or acquire
  - Easy to intersect with a ray (i.e. ray intersects a single quad and not thousands of triangles)
  - Looks convincing when seen head on

Billboard examples

- Specular reflections: 73 billboards
  - Each reflected object for each reflector: 8x9 = 72
  - Ground: 1 billboard

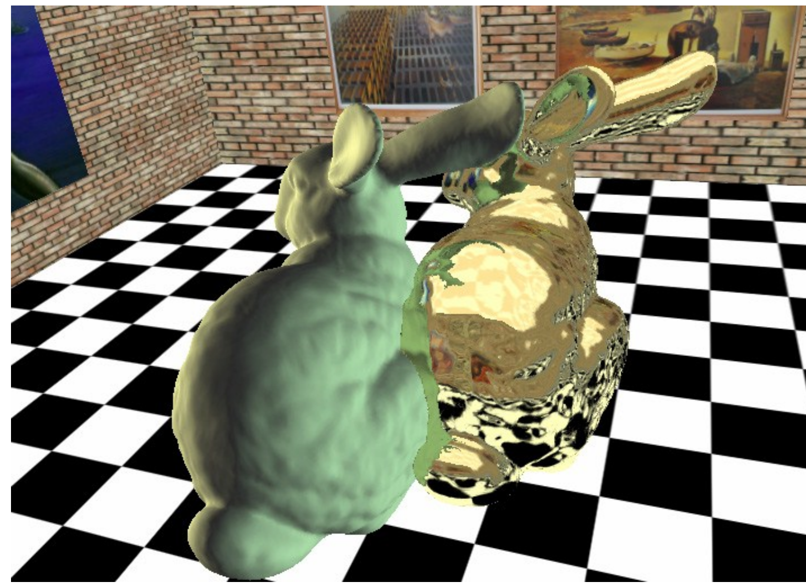




Billboard

- Advantages
  - Easy to render or acquire
  - Easy to intersect with a ray (i.e. ray intersects a single quad and not thousands of triangles)
  - Looks convincing when seen head on
- Disadvantage
  - Looks bad from tangential view directions
  - A single plane of depth

Billboard limitations



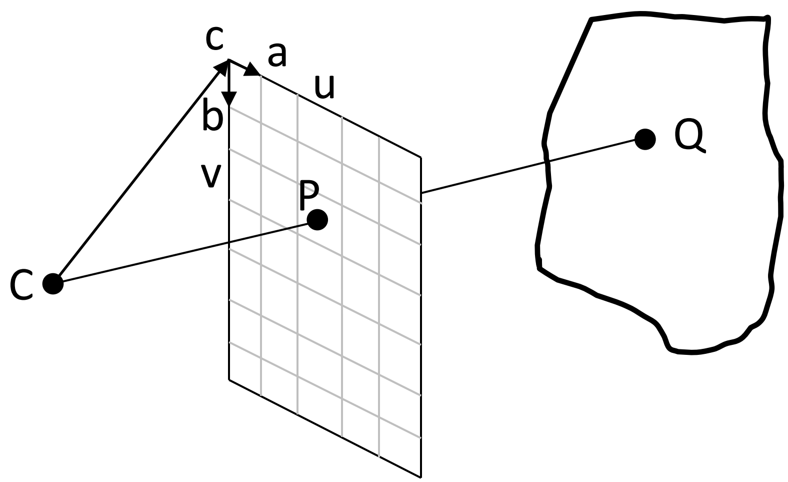
Diffuse bunny cannot be approximated with a single billboard

Depth image

- Definition
  - A conventional image, plus
  - per pixel depth, plus
  - the camera that rendered the image

Depth image

- Geometry approximation
  - Each pixel defines a 3-D point (the closest surface point along the ray through the pixel center)



C- eye (capital C)
c- vector from eye to top left image corner
a- vector with direction given by pixel row and length given by pixel width
b- vector with direction given by pixel column and length given by pixel height
P- center of pixel (u, v)
w- "depth" at pixel P, i.e. CQ/CP
Q- Surface point sampled at P

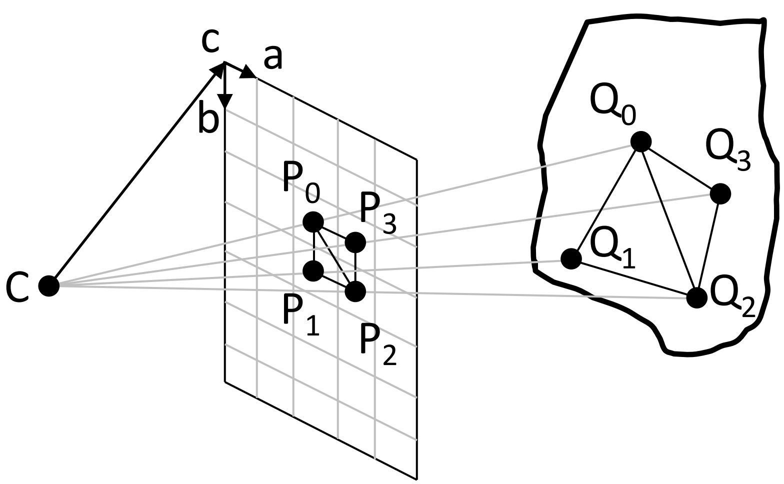
$P = C + au + bv + c$
$Q = C + (au + bv + c)w$

Depth image example

Depth image from reference view (left) and from novel viewpoint (right)

Depth image

- Implicit connectivity
  - Four neighboring pixels can be connected to form two triangles
  - Connectivity information does not need to be stored explicitly due to structure regularity



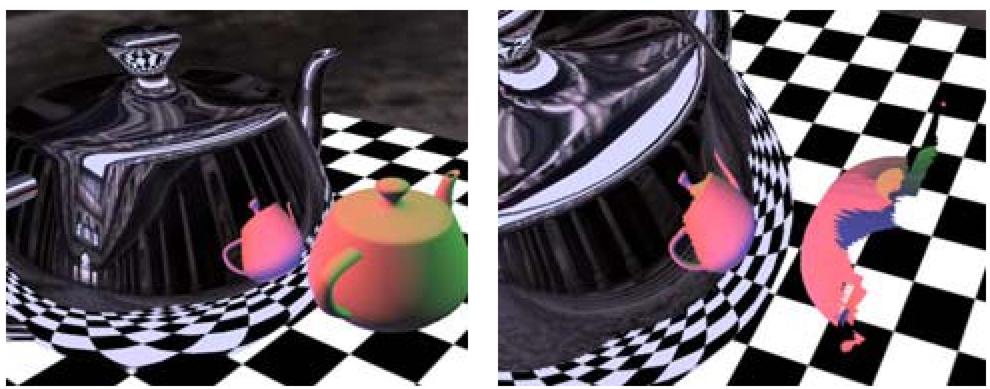
Neighboring pixels P_0, P_1, P_2, and P_3 define two triangles in 3-D, $Q_0Q_1Q_2$ and $Q_2Q_3Q_0$

Depth image

- Advantages
  - Easy to construct for synthetic scenes (just render conventionally and keep z-buffer)
  - Geometry level of detail (LoD) easily controlled through depth image resolution
  - Fast intersection with ray (more on this later)

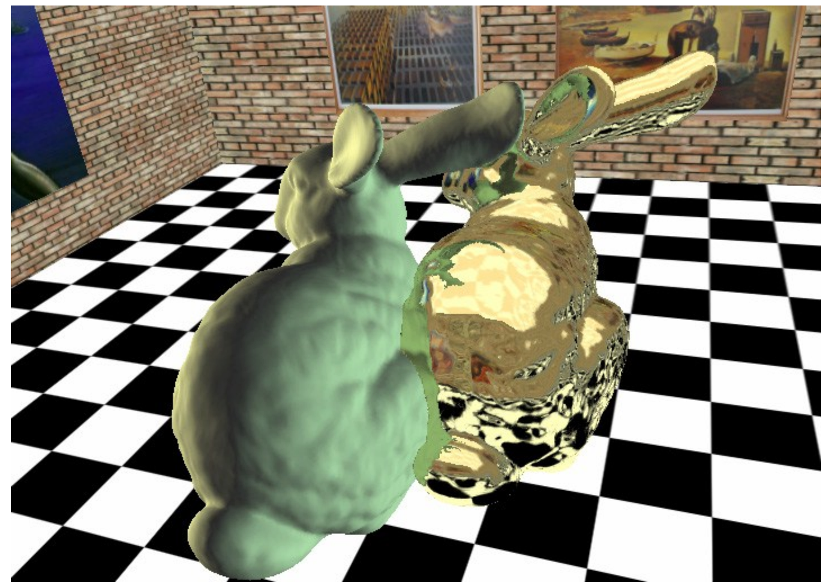
Depth image application

- Specular reflections
  - Geometry of reflected object approximated with depth image rendered from center of reflector



Reflections rendered with depth image impostor (left)
and depth image sample visualization (right)

Depth image application

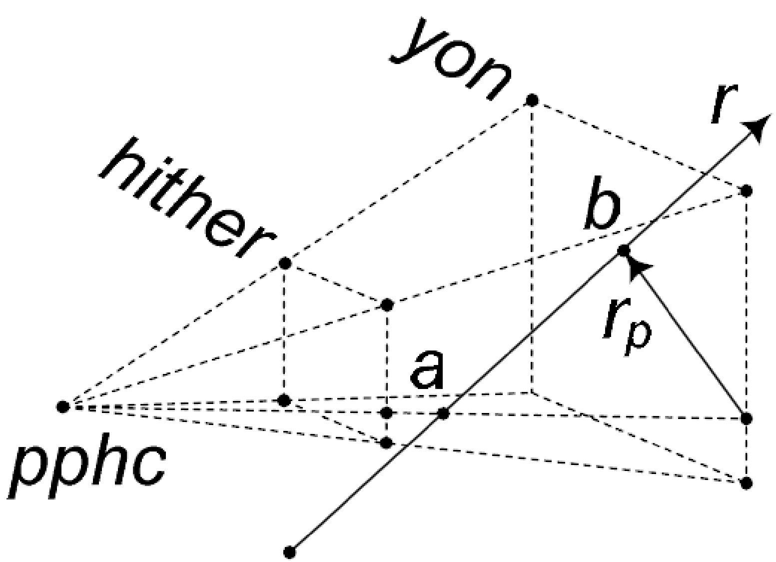


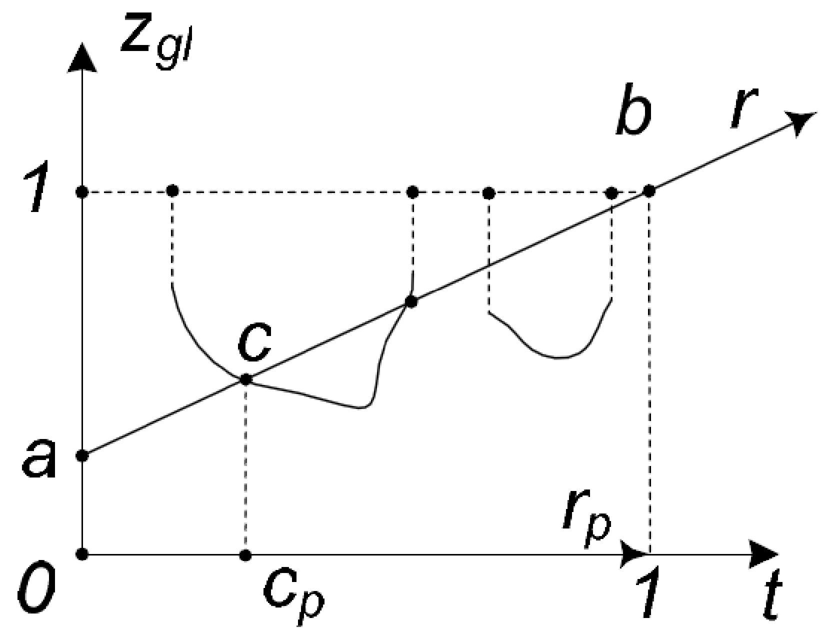
Reflections rendered by approximating the diffuse bunny with a depth image. The depth image models geometry with far higher fidelity than billboards, making the challenging example shown here tractable.

Ray / depth image intersection

- Given a depth image DI of res. wxh and a ray r, find the closest intersection between r and DI
- One could intersect the ray with all triangles defined by neighboring pixels
  - wxhx2 ray / triangle intersections
  - too expensive, not needed
- Project ray onto depth image and only consider depth image pixels under the ray projection
  - *There are at most max(w, h) such pixels*

Ray / depth image intersection





ppch- camera of depth image, with frustum defined by field of view and by near (hither) and far (yon) planes

r- ray intersecting depth image frustum at a&b

r_p- projection of ray onto image (yon) plane

Graph of depth along ray projection ab

z_{gl}- OpenGL depth

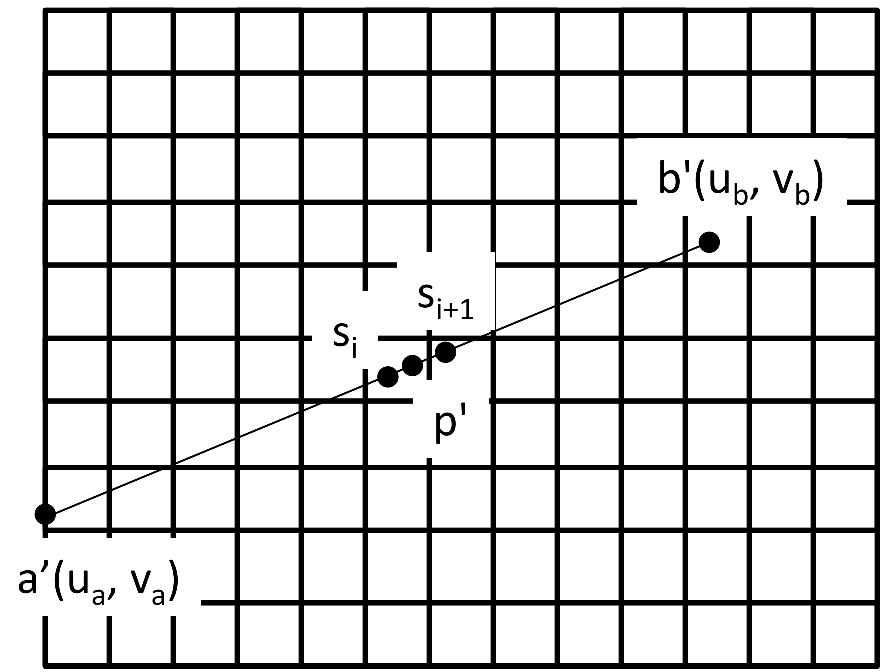
t- parameter along ab

curves- surface sampled by depth image

c- intersection between ray and depth image that is closest to the eye

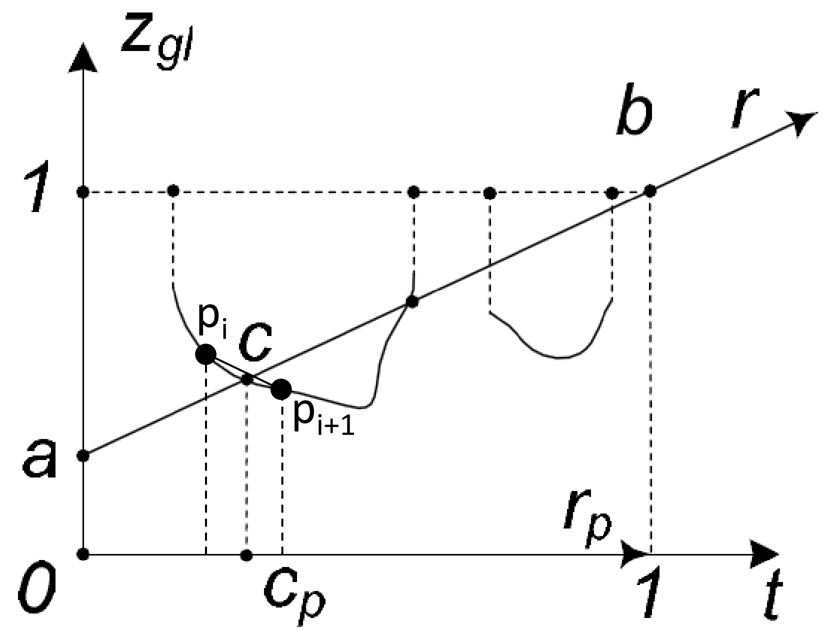
Intersection implementation

- Input
  - Ray direction *r*
  - Depth image *DI*(*ppc, RGB, Z*)
- Output
  - Closest intersection between *r* and *DI*
- Algorithm
  - *(a, b) = ppc.Clip(r);*
  - *a' = ppc.Project(a)*; *b' = ppc.Project(b);*
  - *stepsN* = **max**(**ceil**($|u_a - u_b|$), **ceil**($|v_a - v_b|$));
  - $s_0 = a'$; $p_0 = ppc.Unproject(s_0, Z[s_0])$
  - **for** *i = 0* **to** *stepsN*
    - $s_1 = a' + (b'-a')(i+1)/stepsN$;
    - $p_1 = ppc.Unproject(s_1, Z[s_1])$;
    - **if** ($p = p_0p_1 \cap ab$) **then** *p' = ppc.Project(p);* **return** *RGB[p'];*
    - $s_0 = s_1$; $p_0 = p_1$;
  - **return** *noIntersection*



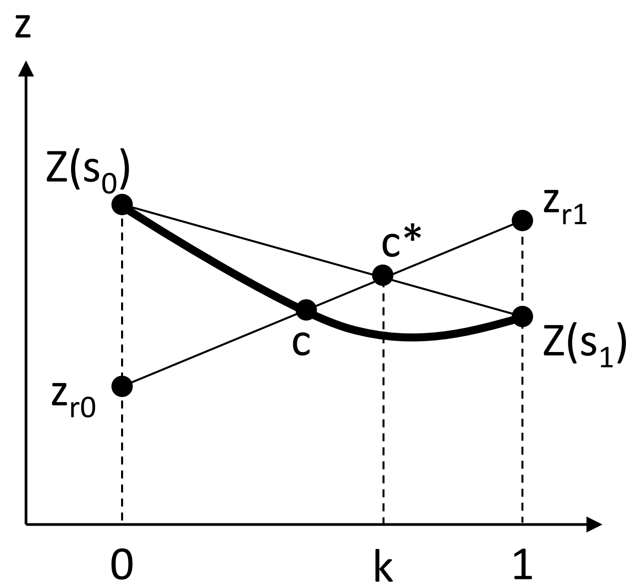
Ray / depth image intersection

Graph of depth along ray projection ab
z_{gl}- OpenGL depth
t- parameter along ab
curves- surface sampled by depth image
c- intersection between ray and depth image that is closest to the eye
p_i, p_{i+1}- segment defined by the depth values at the current and previous steps



Implementation optimization

- Optimized algorithm
  - *(a, b) = ppc.Clip(r);*
  - *a' = ppc.Project(a); b' = ppc.Project(b);*
  - *stepsN* = **max**(**ceil**($|u_a - u_b|$), **ceil**($|v_a - v_b|$));
  - $s_0 = a'$; $z_{r0} = a'.z$;
  - **for** *i = 0* **to** *stepsN*
    - $s_1 = a' + (b'-a')(i+1)/stepsN$;
    - $z_{r1} = a'.z + (b'.z-a'.z)(i+1)/stepsN$;
    - **if** ($k = [(0, z_{r0}), (1, z_{r1})] \cap [(0, Z[s_0]), (1, Z[s_1])]$)
      - **return** $RGB[s_0 + (s_1-s_0)k]$;
    - $s_0 = s_1$; $z_{r0} = z_{r1}$;
  - **return** *noIntersection*
- Faster and simpler to implement
  - Intersection of two 2-D segments
  - No unprojection (to 3-D) and reprojection (to 2-D)



The depth image approximates the true surface (thick line) with segment $[(0, Z[s_0]), (1, Z[s_1])]$.

The intersection c* between the ray and the depth image is computed by intersecting segments $[(0, Z[s_0]), (1, Z[s_1])]$ and $[(0, z_{r0}), (1, z_{r1})]$.