Deep G-Buffers for stable Global Illumination Approximation

Ferit Tohidi Far

February 23, 2019

Content

- Global illumination
  - Pathtracing
  - Visual effects
  - Radiosity
  - Inefficiency
- Traditional rendering
  - Forward rendering
  - Deferred rendering
- Deep G-Buffers)
  - Benefits
  - Generating 2-layer deep g-buffer
  - Global illumination approximation

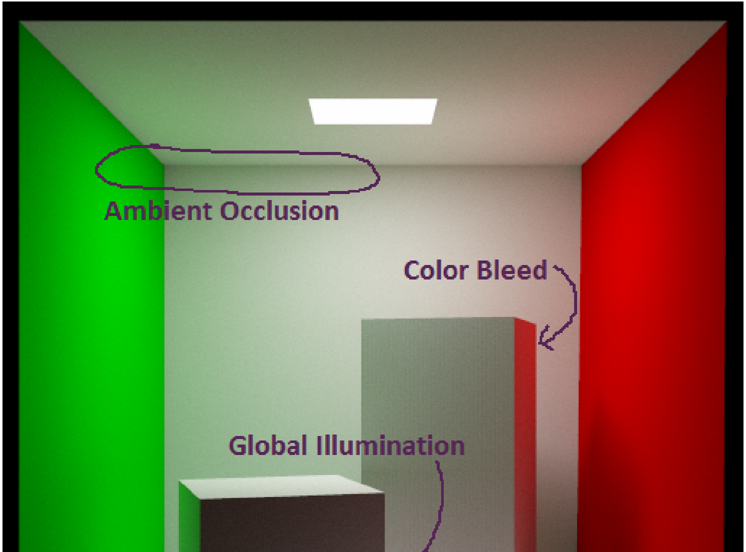
Global illumination

- lighting of a scene
- direct **and indirect** light is considered
- causes visual effects that convey realism
- most popular method is pathtracing

Pathtracing

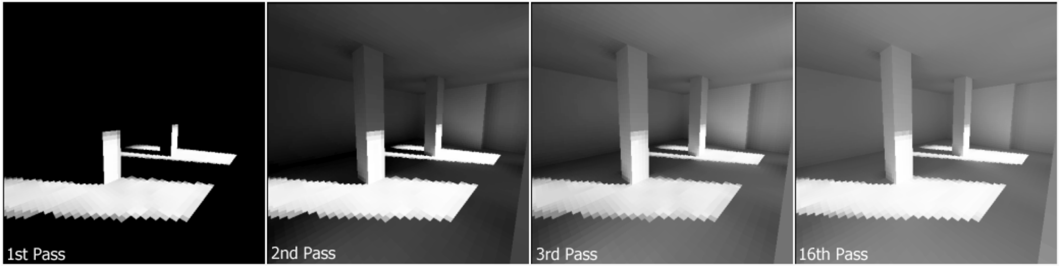
- monte-carlo simulation
- send camera ray through each pixel
- allow ray to reflect **diffusely or specularly**
- trace it back to a light source
- if a light source was hit, the pixel is colored (albedo of hit-object)
- else the pixel is black (in shadow)
- each pixel is sampled thousands of times, then averaged

Visual effects



Radiosity

- scene is divided into patches
- each patch is a light receiver and emitter
- initialize scene with at least one patch that emits non-zero amount of light
- iteratively update receivance and emitance of each patch
- **purely diffuse global illumination**



Inefficiency

- pathtracing
  - computes multiple ray bounces
  - requires thousands of samples (probably more)
- radiosity
  - needs large amount of patches for good results
  - has to be recomputed if an object moves
- both cannot be computed in real-time (yet)

Traditional rendering

- rasterization
- way easier to compute than raytracing
- interactive framerates
- trade-off: not as realistic
- requires techniques to simulate visual effects

Forward rendering

- computes lighting in a single pass:
    - for each fragment compute lighting
    - do z-test
    - render to frame-buffer or discard accordingly
    - render frame-buffer to screen
- computes lighting for fragment regardless if visible or not

Deferred rendering

- computes geometry (stored in g-buffer) in first pass:
  - albedo-buffer
  - normal-buffer
  - z-buffer
- render g-buffer to texture-buffer
- compute lighting in second pass:
  - read frontmost scene geometry from g-buffer
  - compute lighting
  - render to screen
- only computes lighting for visible fragments

Deep G-Buffers

- generate 2-layer deep g-buffer with depth-peeling or oracle
- enforce minimum depth separation
- consider second layer for visual effects

Generating a 2-layer deep g-buffer (depth-peeling)

- **depth-peeling method**
- collect first layer g-buffer as usual
- compute second layer g-buffer by stripping the first layer
- takes two passes over scene geometry

Generating a 2-layer deep g-buffer (depth-peeling)

- **depth-peeling method**
- collect first layer g-buffer as usual
- compute second layer g-buffer by stripping the first layer
- takes two passes over scene geometry
- instead use oracle and do it in a single pass!

Generating a 2-layer deep g-buffer

- **use some oracle to predict first layer z-buffer**
- remember that we are running some simulation/animation
- frames are computed per time-step
- after a time-step, the object locations won't change **that much**
- we even have knowledge of position and velocity updates
- exploit this by recycling information from the previous frame and adjusting it a little
- 4 different variants available

Generating a 2-layer deep g-buffer (previous variant)

- **previous variant**
- recycle first layer z-buffer of previous frame
- the smaller the position updates, the smaller the error
- even then, errors would only appear in the second layer (invisible unless transparent)
- does not guarantee minimum separation

Generating a 2-layer deep g-buffer (delay variant)

- **delay variant**
- introduce a frame of latency
- frame and animation/simulation are out of sync by one frame
- use first layer z-buffer of precomputed latency frame
- drawback: one frame of latency

Generating a 2-layer deep g-buffer (predict variant)

- **predict variant**
- use velocities from animation/simulation
- predict position updates of objects
-

Generating a 2-layer deep g-buffer (reproject variant)

- **reproject variant**
- performs minimum separation test against previous frame's first layer z-buffer
- visibility test done using previous z-buffer ("in the past")
- same source of error as predict variant, but not as bad (velocities are perfect)
- delivers most stables performance out of the 4 variants

Global illumination approximation using Deep G-Buffers

- compute ambient occlusion
- compute some radiosity steps
- compute screen space reflections
- apply direct and ambient light

Ambient occlusion with Deep G-Buffers

-

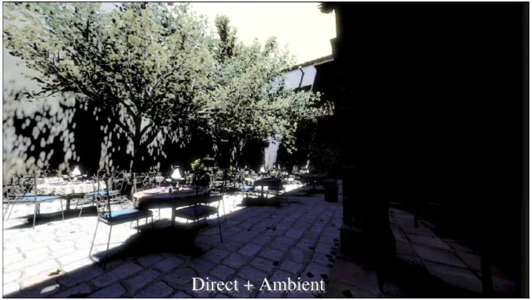
Reflections with Deep G-Buffers

-

Radiosity with Deep G-Buffers

- temporal filtering for undersampling noise reduction

Results



- 1920x1080 resolution
- rendered using NVIDIA GeForce 980
- in 10.8ms (92 FPS)
- looks good