Cheap and Dirty Irradiance Volumes for Real-Time Dynamic Scenes

Rune Vendler

VGC

Agenda

- Introduction
- What are irradiance volumes?
- Dynamic irradiance volumes using iterative flow
- Mapping the algorithm to modern GPUs
- Ideas and extensions

Introduction

- Worked professionally with real-time graphics for about 12 years
  - Interactivision, Lionhead, VGC
- Interest is primarily pragmatic
  - As long as we can deliver beautiful images at a steady framerate and with reasonable development cost, nothing else really matters

VGC

Part I:
What are irradiance volumes?

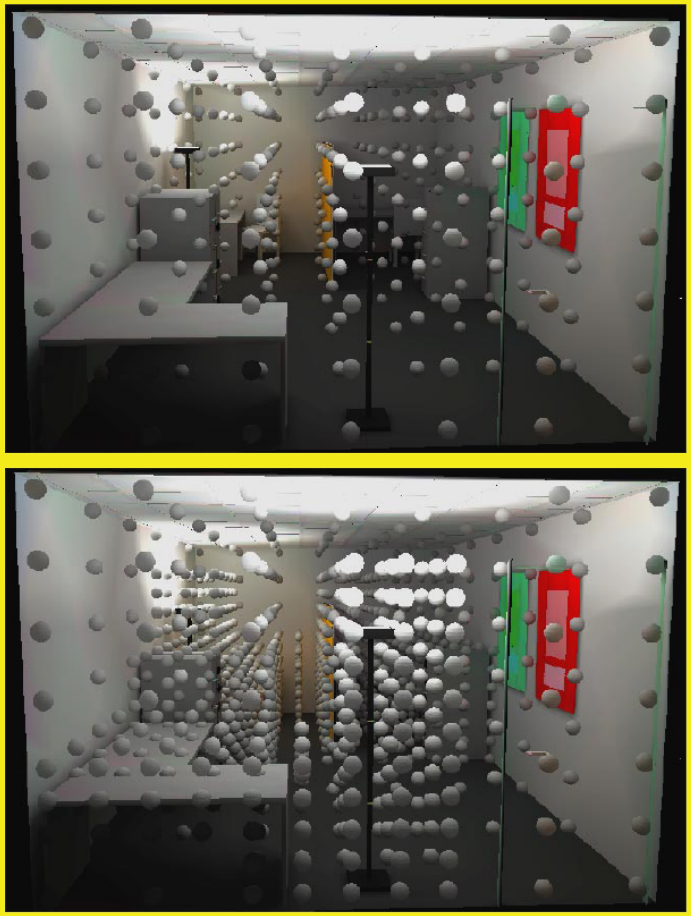
- Irradiance samples and volumes
- Examples from games

An irradiance sample

- A single irradiance sample...
  - Stores incoming light for a particular point in space
    - I.e. the light that flows through this point
  - Is often parameterised by direction
    - I.e. We can ask: how much light comes from *that* direction
    - Stored in e.g. an environment map or spherical harmonics

Irradiance volumes

- With a set of samples, we can approximate a volume
  - We can look up any point by interpolating the actual samples
- Many ways to compute samples
  - Often time consuming to compute, but trivial to access afterwards



Examples from games

- Quake 3: Ambient/directional color, direction
- Half-Life 2: ”Ambient cube” – 6 ambient colors
- Little Big Planet: Diffuse color (view-aligned volumes)

Different goals, different solutions!

Part II:
Dynamic irradiance volumes

- Basic idea
- 2D proof-of-concept
- 3D implementation
  - Bare bones
  - Dynamic objects

Problem domain

- Goals
  - Dynamic environment
  - Dynamic lighting
  - Must encapsulate all lighting in the scene
- Constraints
  - World changing slowly
  - Relatively small environment
  - No programmable GPU
    - Must run on a Nintendo Wii, i.e. relatively slow CPU + fixed function GPU

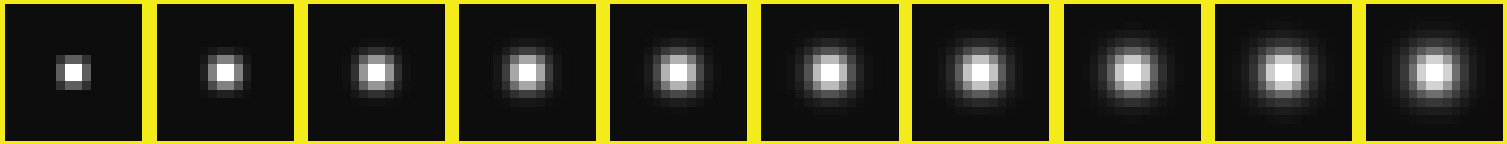
Iterative flow

- Fundamental idea:
  - Spread the work across many frames
    - Simulate light moving through the scene interatively, rather than instantaneously
  - With each interation, the result should converge on
    - A stable state...
    - ...that looks good
  - Simplest possible representation:
    - Regular grid, either solid or open space
    - How light is this cell: no directional information
  - Each iteration flows light one cell

VGC

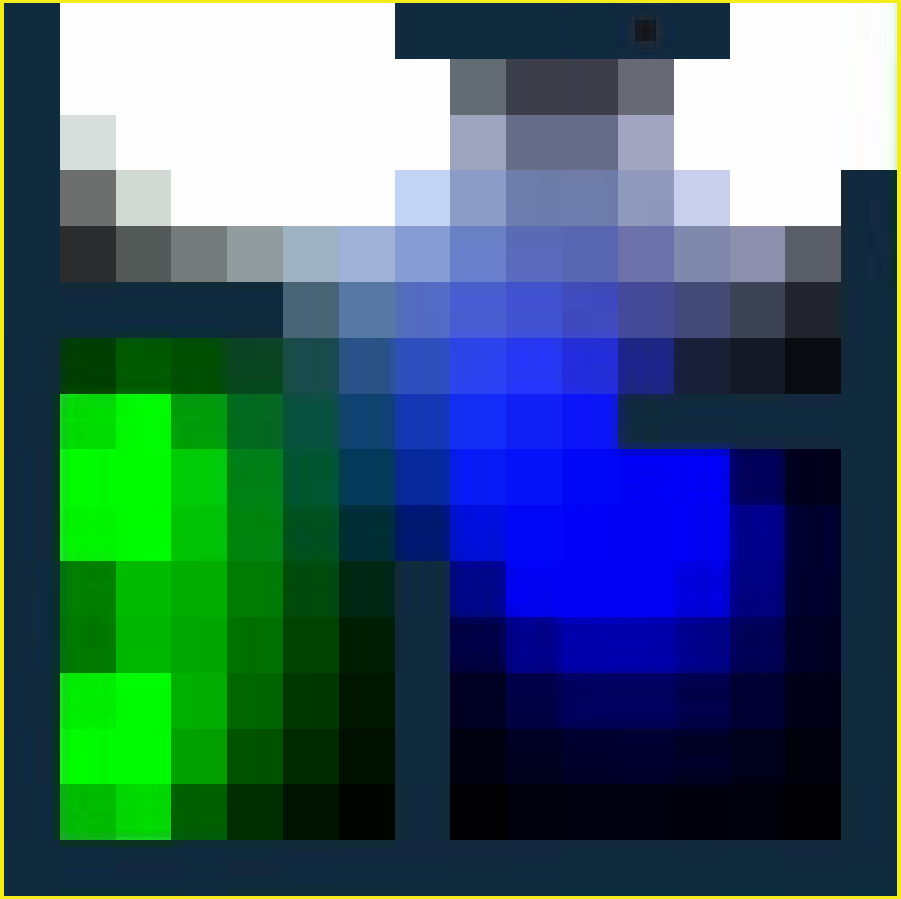
One iteration

- Transform volume V to V’
  - All reads from V and companion data, all writes to V’
- For each cell C, to calculate light L
  - Gather light:
    - If C is soild space, L = 0
    - else L = average of neighbor cells
      - (up+down+left+right)/4 – or diagonals too
  - Emit light
    - If cell is emitter, L += emitted light



VGC

One-hour 2D proof-of-concept demo



VGC

It's a blur

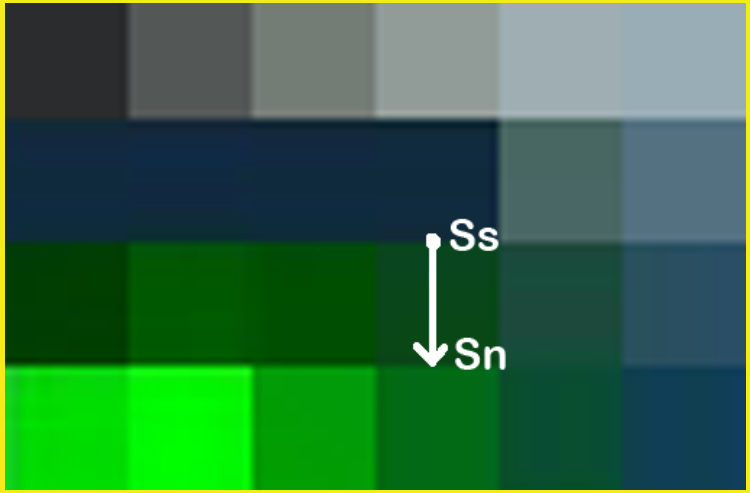
- The core algorithm is just a blur algorithm
  - Blur kernel area must be small enough to never sample the wrong side of a wall
- Then we force solid space to 0, and add light where we want emitters
- Needs high dynamic range to carry light any real distance

Extending to 3D

- Trivial extensions
  - 3D grid, 3D blur kernel, trilinear lookup
- Other
  - Taking normals into consideration
  - Sampling the volume
  - Handling dynamic objects
  - Fractional occlusion

Taking normals into consideration

- We need to derive directional information somehow
- Intuition: comparing two cells next to each other:
  - If A is brighter than B, that suggests light is flowing from A to B
- Now, take two samples: at surface, Ss, and at surface+normal, Sn
  - Grad = Sn – Ss
  - Light = Ss + F * Grad
  - F = fudge factor



VGC

Sampling the volume

- We need to sample the volume and store the results for rendering somewhere
  - Vertex colors
  - Lightmaps
- Alternatively, we can sample the volume per pixel while rendering
  - Volume must be accessible from GPU
  - Requires a bit of programmability

Handling dynamic objects

- For dynamic objects, this kind of detailed sampling might not be possible
  - Too expensive
  - Not compatible with hardware
- Instead, we reconstruct the lighting environment around the object, and represent it somehow that the fixed function hardware likes
  - Fixed function lights!
- Let's surround the object with directional lights that represent the light environment

Fractional occlusion

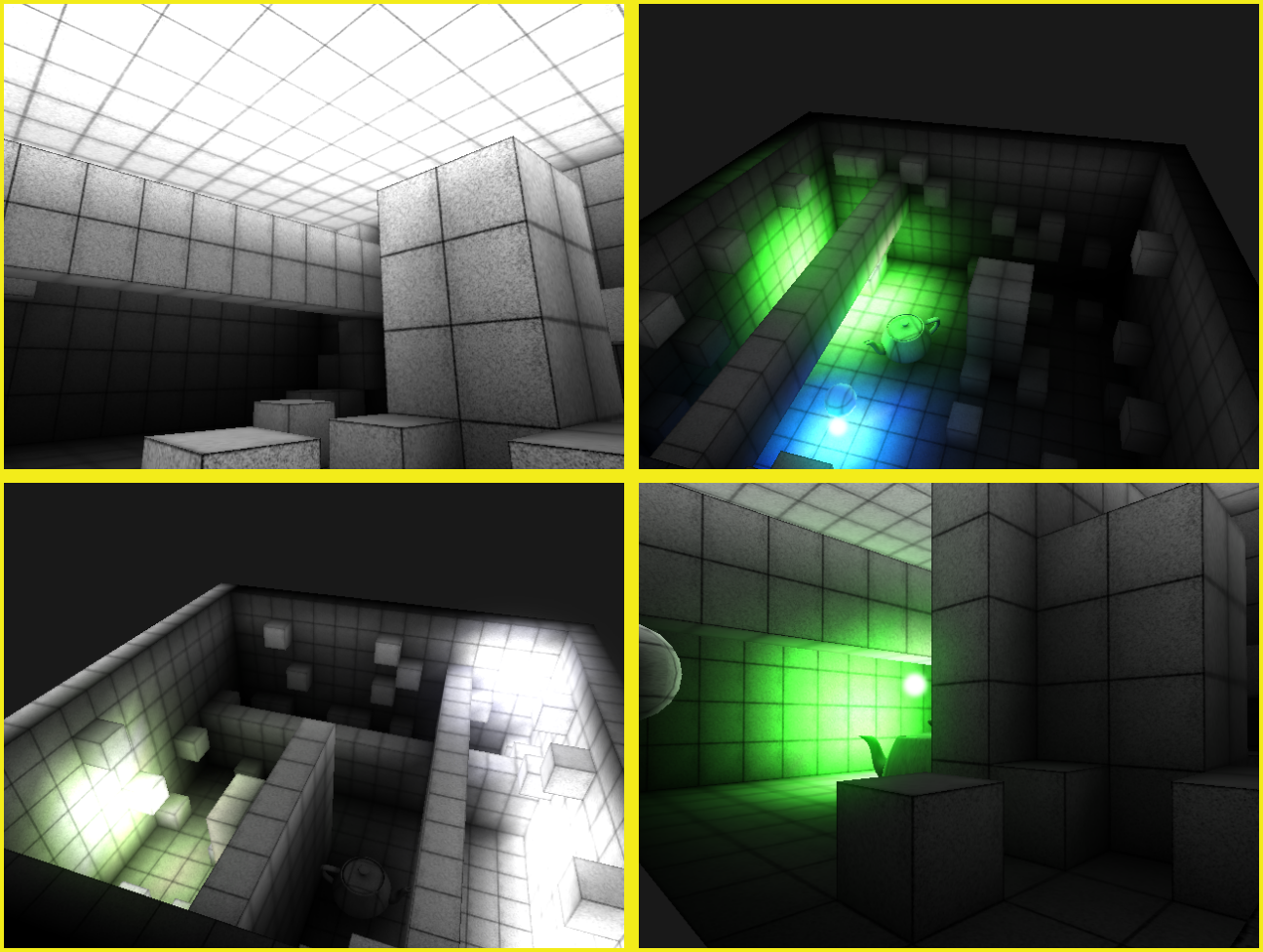
- Occlusion doesn't have to be binary
  - I.e. 0% or 100% light gets through
- We can occlude (darken) by any percentage we like
- Can potentially be used to express semi-occluded space
  - Dynamic objects
  - Transparent scenery

VGC

Speed

- Interesting aspects:
  - Cost is linear in the number of cells
    - I.e. Size of space
  - Cost is constant in the number of lights
    - We can represent big area lights by just sticking in lots of lights in cells next to each other
  - Most (if not all) operations are really simple
    - No branches necessary

Demo!



Part III:
Mapping to modern hardware

- Sampling the volume
- Moving the blur to the GPU
- Improvements and variations
  - Directional component on occlusion
  - Light scratchpad
  - Combining with direct lighting

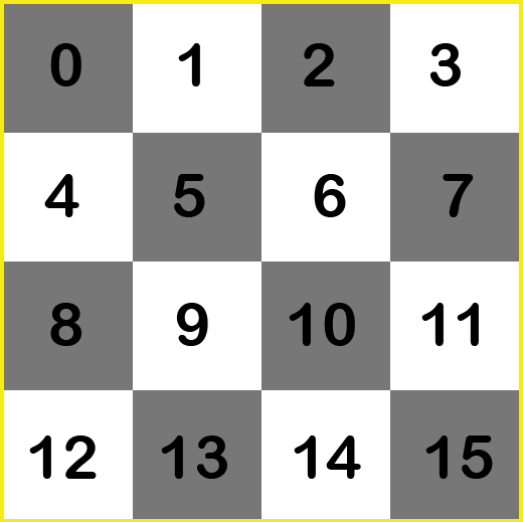
VGC

Sampling the volume

- Sample in pixel shader
  - Need access to world position and normal
  - No need to differentiate between static and dynamic objects
- Trivial to stick the volume in a 3D texture, but this might not be the best choice

Moving the blur to the GPU

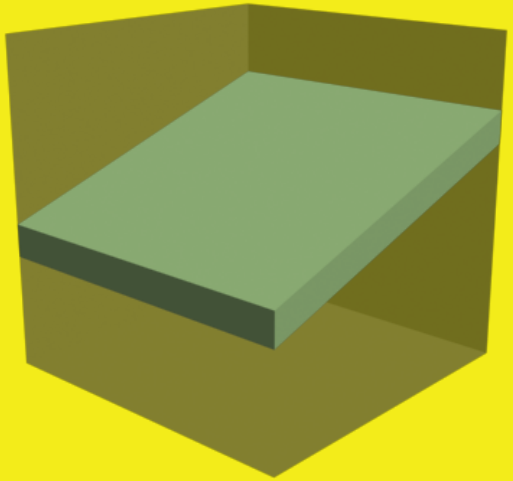
- Volume as render target
- Can't render to 3D texture
  - Instead, we'll store the 3D texture as slices on a 2D texture
  - In the shader, we convert from 3d coordinates to 2d coordinates on the texture
- Light and occlusion stored in other textures, or added in another pass
  - E.g. as point primitives



VGC

Directional component on occlusion

- Instead of a single occlusion value, we can store one per axis
  - This changes the blur kernel per pixel, but allows more directional control over the light flow
- Find coverage e.g. through rasterisation



Left – Right Front – Back Top - Bottom

VGC

Light scratchpad

- View the volume as a 3D scratchpad for light in the scene
  - Write anything you want into it
- Can be kept as a separate map that is created from scratch every frame

Combining with direct lighting

- Light scene with traditional direct lights/shadows, then add in contribution from irradiance volume
- Instead of putting discrete lights into the irradiance volume, add every surface and its light to the volume
  - I.e. every surface casts light based on what it received from the direct lighting
  - Effectively allows light to ”bounce”

VGC

Getting away from cubes

- The cubes are really only the occlusion representation of the scene
- Many looks lend themselves well to cube-based occlusion, especially for indirect lighting

Conclusion

- There are many ways to make use of irradiance volumes
  - It's yet another good tool in the toolbox
  - Know your problem domain!
- Thanks!
  - Kasper Høy Nielsen, IO Interactive
  - Alex Evans, Media Molecule

Questions?

rune@vendlergc.com