Lightcuts in Blender

Davide Vercelli
Scuola Superiore S. Anna
Pisa, Italy
Google Summer Of Code

- encourages students to participate in open source development
- students propose an improvement over existing open source software
- core developers of participating organizations review the proposals
- students code over the summer, funded by Google
- most importantly, they get a Google T-Shirt!
What I'm going to talk about

- The “Lightcuts” Algorithm
- Implementing Lightcuts in Blender
- Integration issues
- Possible future directions
The Lightcuts Algorithm
Shading from a single point light

- mathematically easy
- all quantities are definite
Shading from a single point light

- hard shadows
- unrealistic lighting
Shading from an area light

- it's not possible to handle it analytically in the general case
- quantities vary throughout the area

\[ s \]
Shading from an area light

- closed formula possible for simple shapes (e.g. rectangle) and ignoring visibility
Numerical approach

- use MonteCarlo sampling
  - a way to compute approximate solutions to this kind of problems (integrals)
  - direction taken by the mainstream rendering research
  - in Blender: AO, IBL, ecc.
“Many point lights” approach

- approximate a light source by placing a large number of point lights
- flexible possibility
  - unifies different kinds of lighting
- a big number of lights is required to have good results
- rendering cost grows proportionally to the number of point lights used
Introducing Lightcuts

- developed at Cornell University by B. Walter et al. (Siggraph 2005)

- objectives:
  - no “smart” reduction of point lights: use as many as you want
  - sublinear rendering cost
    - if k lights render in T time
      2 * k lights must render in << 2 * T time
Lightcuts: core idea

- let's say you have two "similar" point lights
  - take intensity, position and direction into account
Lightcuts: core idea

- use a single “representative” light instead
  - increase its intensity accordingly
Lightcuts: core idea

- The difference would be unnoticeable for most, but not all, pixels.
Lightcuts: core idea

- at each pixel make a choice between:
  - calculating the contributions of every single light
  - render a single representative light instead

- choose the more accurate computation only when the error is too noticeable
“Noticeable” errors

• take human perception into account
  – we are sensitive to signal ratios more than absolute values (Weber's Law)

• a pixel intensity which is wrong by a fixed percentage w.r.t. the correct value goes unnoticed provided the percentage is low enough (~2%)
From theory to practice

- how to extend the idea of representative lights to *thousands* of lights?
- how to handle that efficiently?
  - it has to be faster than evaluating all lights!
- how can we determine upfront if our approximation is acceptable for the current pixel?
  - if we knew the correct value for the current pixel there would be no need to find an approximate value!
The light tree

- a global, scene wide binary tree
- the leaves are the actual lights
- inner nodes are cluster representatives
The lightcut

- it's the set of lights we actually evaluate at a given pixel
- most of them will be cluster representatives but some of them could be actual lights
The lightcut

- a path from the root to an actual light contains only a single node of the cut
- each cut represents a different partitioning of the actual lights into clusters
How to select the cut

- we start from the root node
How to select the cut

- we compute “temporary” shading and check if it exceeds an error metric
How to select the cut

- if it does, we exclude the node from the cut and we add its two children
How to select the cut

- now we look for the node in the cut giving the biggest contribution to the final shading
How to select the cut

- at each node we compute temporary shading and we go deeper in the tree if it exceeds an error metric
How to select the cut

- if it exceeds our error metric, again: we remove it from the cut and we add its two children
How to select the cut

- this process continues until we are satisfied with our current cut
How to select the cut

- this process continues until we are satisfied with our current cut
How to select the cut

- this process continues until we are satisfied with our current cut
How to select the cut

- this process continues until we are satisfied with our current cut
How to select the cut

- this process continues until we are satisfied with our current cut
How to compute the error estimation

• compare with the maximum possible value you can get from the cluster

• if it's under a fixed percentage of the current shading estimation that's true also for the actual value
  – so the approximate value is acceptable
  – no need to refine the cut here

• but the bounding has to be both **tight** and **cheap**
How to compute the error estimation

- the contribution of a single light has three components:
  - geometry
  - material
  - visibility

- we can bound them separately
Bounding the geometric factor

- for omnidirectional lights (Lamp) we bound the falloff
- each cluster maintains a bounding box so that it's easy to find a minimum distance
- in Blender we have to consider the various (and little known) falloff functions

$S$  

minimum distance
Bounding geometry

- for oriented lights (180° Spot) we bound both the falloff and the direction.
- If you are outside the bounding cone, you can compute the minimum directional attenuation that you get.
Bounding geometry

- for omnidirectional lights (Sun) there's no bounding
Bounding materials

- a delicate part
- a different bounding function for each shader
- fairly easy for diffuse shaders
- less trivial for specular ones

potentially large contribution

cluster representative with small lambertian contribution
Bounding visibility

- pretty hard in the general case
- the original paper left it unbounded (1)
Main benefits

• as complexity is added to the lighting rendering times don't grow proportionally
  – artists are thus encouraged to make extensive use of area lights, environment lighting, etc.

• unify in a single strategy the handling of lights coming from different sources so you can easily add:
  – textured area lights
  – meshlights
  – some forms of indirect lighting
Minor benefits

- by raising the error threshold, faster preview renderings can be generated in order to adjust lighting
  - a bit scene dependent though
Implementing Lightcuts in Blender
Supported point light sources

- actual point lights
- area lights
  - meshlights
- environment lighting
- Instant Radiosity
Actual point lights

- Lamp
- Sun
- Spot
  - only 180°
- they give hard, single sampled shadows
  - they are meant to be used in big numbers
  - of course you may want a single sun, though
Area lights

- set the global number of samples then distributed by area
- new feature: textured area lights
Meshlights

• Materials with Emit become actual lights
Meshlights

scene by bullx
Environment lighting

- similar to the AO+sky texture counterpart
Instant Radiosity

- simulate **indirect lighting** by placing a big number of point lights where direct lighting hits the scene
- research in this field tried hard to keep the number of Virtual Point Lights low while maintaining the indirect lighting effect
- thanks to Lightcuts we can use a big number of VPLs without worrying
lightcuts 20000 lights
Supported shaders

- all diffuse ones
- phong specular
- other supported features
  - sphere, bias, etc.
Other supported features

- baking
- SSS
Current issues

- noticeable flickering in animation :(  
  - bugs?
  - inherent limitations esp. with IR
  - maybe precision issues as well?
- rendering times are good but not compelling
  - heavily depends on raytracing performance
  - see future directions
Integration issues
User Interface

- someone complaining about messy UIs?
- most sliders and buttons are only there for debugging purposes though...
Unsupportable features

- example: custom curve falloff for lights
  - how can that be boundable in the general case?
- true for many other “unrealistic” features
- another serious problem is with “node materials”
  - how to bound them in the general case?
Unintuitive performance

- performance is not impressive if you have too few lights!
- avoid large occluded areas
  - slow because of the relative error metric
  - it's actually an encouragement to use environment lighting or indirect lighting
- occlusion, more than lighting complexity, affects rendering times
  - i.e. environment lighting from a small window is relatively slow
How to integrate Lightcuts in the workflow?

- ideally: a switch that changes things only “behind the curtains”
  - but performance is more difficult to predict
  - lighting rigs are thought out differently when indirect lighting is taken into account
  - not a perfect correspondence i.e. in area lights
Possible future directions
Reconstruction cuts

- adaptive screen space interpolation
  - you run the lightcuts algorithm every few pixels
  - for pixels in between, the lightcuts *themselves* get interpolated not the pixel values
  - no interpolation through discontinuities
- should improve rendering times and image smoothness considerably
- how does this fit with Blender's design?
Better indirect lighting

- current implementation mostly a proof of concept
- state of the art seems to be Metropolis Instant Radiosity
  - but please note that the paper does mention flickering in animation!
Multidimensional Lightcuts

- extend the same principle (adaptive per-pixel clustering, etc.) to other kinds of sampling
  - motion blur
  - depth of field
  - sub-surface scattering
- high quality images
- would it fit with the workflow and the codebase?
Thank you for listening!

*Ant thanks to:*
Blender Foundation
Google
Kent Mein (my mentor)
testers on kino3d,
blenderartists, #blendercoders
...and let's take the opportunity to thank also the other 2008 “gsoccers”:

Nicholas Bishop  
Maxime Curioni  
André Pinto  
Ian Thompson

... and their mentors of course!