The idea
The idea

• Use Radiance to “bake” light maps of models
• Light maps can be used for animations and games
• Maps can be used to calculate values over complex surfaces
• Maps can also be used to bake geometry
Are you sure it’s something new?
**Lightmaps from HDR problems**, 1st Radiance Workshop
Bernard Spanlang,
VECG Group University College of London, 2003

---

**Possible Improvements**

- Maintain connectivity of 3D triangles for 2D UV texture coordinates
- Where not possible add extra pixels for interpolation (Sand pixels)
- Triangle areas reflected in texture size
- Coplanar surfaces represented by one lightmap element
- Packing lightmap elements

---

[Diagram showing the workflow of lightmaps generation and rendering]
Multidisciplinary 3D Spatialisation Simulation, Francesco Anselmo, Arup internal research, 2006

Python script + Blender GUI
Number of polygons and resolution are limited
UV mapping is not imported from the model
So?

- Previous methods do not use existing UV sets
- New UV sets are instead created in the process
- Speed of the process suffers due to UV creation
- UV sets cannot be exchanged with the CG artist
A refined approach:

- The same UV set of the model is used for baking
- Dedicated modules (import/calculate/ process...)
- Simple multicore speedup
The method in 7 steps
How does it work?

1. Parse 3D files and import UV mapping and points
2. Reconstruct the UV transformation matrix
3. Generate a grid of points in UV space (texture pixels)
4. Find the 3D location of each UV texture pixel
5. Render each pixel separately and in parallel
6. Filter image for seams
7. Save all data together in a single image file
What is required:

- Octave / Matlab for fast matrix operations without the complication of C++
- Radiance
- A 3d model in .obj format with UV mapping
1 read the UV and 3D coordinates from .obj files
1 Anatomy of an .obj file

# WaveFront *.obj file (generated by CINEMA 4D)
g sea
usemtl sea
v -1215.676758 0 1307.318848
v 1784.323242 0 1307.318848
v -1215.676758 0 -1692.681152
v 1784.323242 0 -1692.681152

vt 0 0 0
vt 1 0 0
vt 0 1 0
vt 1 1 0

f 2/2 4/4 3/3 1/1

Example .obj file.
# WaveFront *.obj file (generated by CINEMA 4D)
g sea
usemtl sea
v -1215.676758 0 1307.318848
v 1784.323242 0 1307.318848
v -1215.676758 0 -1692.681152
v 1784.323242 0 -1692.681152
vt 0 0 0
vt 1 0 0
vt 0 1 0
vt 1 1 0
f 2/2 4/4 3/3 1/1
Anatomy of an .obj file

# WaveFront *.obj file (generated by CINEMA 4D)
g sea
usemtl sea
v -1215.676758 0 1307.318848
v 1784.323242 0 1307.318848
v -1215.676758 0 -1692.681152
v 1784.323242 0 -1692.681152

vt 0 0 0
vt 1 0 0
vt 0 1 0
vt 1 1 0

f 2/2 4/4 3/3 1/1
1

Anatomy of an .obj file

# WaveFront *.obj file (generated by CINEMA 4D)
g sea
usemtl sea
v -1215.676758 0 1307.318848
v 1784.323242 0 1307.318848
v -1215.676758 0 -1692.681152
v 1784.323242 0 -1692.681152

vt 0 0 0
vt 1 0 0
vt 0 1 0
vt 1 1 0

f 2/2 4/4 3/3 1/1

Polygon connections and UV mapping.

ArupLighting
Anatomy of an `.obj` file

# WaveFront *.obj file (generated by CINEMA 4D)

g sea
usemtl sea

1 v -1215.676758 0 1307.318848
2 v 1784.323242 0 1307.318848
3 v -1215.676758 0 -1692.681152
4 v 1784.323242 0 -1692.681152

1 vt 0 0 0
2 vt 1 0 0
3 vt 0 1 0
4 vt 1 1 0

f 2/2 4/4 3/3 1/1
We can read an .obj file and find, for each triangle in 3D, the corresponding triangle in UV.
Acknowledgement!
The parser proposed is based on the work of William Harwin, University Reading, 2006

See here:
http://www.mathworks.com/matlabcentral/fileexchange/10223
2
derive the transformation matrix from known UV and 3D points
This is a 3D model with textures

This is a UV map
The same triangle $T$ has different vertex coordinates in the two vector spaces.
Is there a way to relate the corresponding vertex coordinates between the UV and 3D planes?
Yes, all we need is to find the affine transformation between UV and 3D spaces.

(http://en.wikipedia.org/wiki/Affine_transformation)
Knowing $M$, it is possible to convert $P$ to $Q$ or $Q$ to $P$. 

$P = M \cdot Q$
$P(x,y,z)$

$Q(U,V)$

$P = M Q$
Basically it is possible to convert point in the UV plane to the corresponding 3D points...

...so that I can rtrace each location in 3D and get the value of the texture...
For details and an extensive how-to find the $M$ matrix, try this link:

http://news.povray.org/povray.general/thread/%3Cweb.442a6fe16260549766ffc7a50@news.povray.org%3E/
3
generate a grid of points in UV space, convert in 3D space
Once the *affine transformation* $M$ is found this 3rd step is pretty much just a matrix multiplication...

Note that each distinct polygon may has a different $M$!
Generate a grid of points in UV
Reduce the grid to the points inside the triangle.
Transform from UV to 3D using the affine transformation matrix $M$
...now we offset the points from the 3D polygon in the normal direction.

Why? Because we need to rtrace towards the polygon to see it!
Offset points normally.
Compose the final calculation grid, including the reverse normal:

$$[P_x \ P_y \ P_z \ -N_x \ -N_y \ -N_z]$$
Repeat 2 and 3 for each triangle in the file.
The affine transformation $M$ may be different for different polygons, therefore we need to evaluate it for each polygon separately...
4
Save a final grid including all points.
Once all polygons have been converted we can finally save a single file for the main rtrace calculation.
Save a final grid including all points.
Or we can split it in several files to enable a crude, but effective, multicore approach...
Divide the file according to the number of cores, for example 2

Grid file with all points

Core 1

Core 2
See here for details on how to split a grid:

http://web.mac.com/geotrupes/iWeb/Main%20site/RadBlog/E549E7F4-6DA2-4D78-8F91-74A4691ED86A.html
5
render with \textit{rtrace}
Use & and wait to run a number of rtrace processes in parallel.
```bash
rtrace -h- model.oct < grid1.grd > grid1.data &
rtrace -h- model.oct < grid2.grd > grid2.data &
wait
```

The script continues only when all the calculations have been completed.
6

Filter seams

Arup Lighting
If resolution is low or mapping non-optimal we could have some empty (black) pixels on the edges of polygons.
Seams and filtering

Problem is mitigated by increasing resolution but never completely resolved...
Seams and filtering 6

Gaps need to be filled
Seams and filtering

Value = max(what is around)
Seams and filtering

There is a lot of bleeding...
Seams and filtering

But once the data is mapped we can only see what is on the polygon...
assemble back in a single image using $p_{value}$
For instance we could use:

```
pvalue -r -o -h -H -da -x 512 -y 512 tex.dat > tex.pic
```
Action!
Complicate geometry...
Complicate geometry...
Post process and animations...
Post process and animations...
Realtime demo...

See here:
Thanks!