Parallel Ray Tracing

See T. Plachetka: Event-Driven Message Passing and Parallel Simulation of Global Illumination, Chapter 4
Global illumination [Kajiya 1986]

Object geometry (set of surface points $x$)
Object materials: $BSDF(x, ω', ω)$
Light sources: $L_e(x, ω)$
Measuring sensors in camera: $W_e(x, ω')$

$RT(x, ω)$ denotes the ray tracing operation which returns the first surface point from point $x$ in direction $ω$.

\[
L(x, ω) = L^e(x, ω) + \int_{Ω} BSDF(x, ω', ω) L(RT(x, -ω'), ω') \cos(θ') dω'
\]

\[
W(x, ω') = W^e(x, ω') + \int_{Ω} BSDF(x, ω', ω) W(RT(x, ω), ω) \cos(θ) dω
\]

Radiance $L(x, ω)$ at least for all points $x$ and directions $ω$ which contribute to the response of some camera sensor.

Potential $W(x, ω)$ at least for all light surface points $x$ and all directions $ω$. 
Global illumination [Kajiya 1986]

\[
L(x, \omega) = \Re\left( L_{\text{direct}}(x, \omega'), L_{\text{reflected}}(x, \omega'), L_{\text{refracted}}(x, \omega'), L_{\text{ambient}}(x) \right)
= m_{\text{diff}} \int L_e(RT(x, -\omega_{\text{light}}), \omega_{\text{light}}) d\omega_{\text{light}}
+ m_{\text{spec}} L(RT(x, -\omega_{\text{spec}}), \omega_{\text{spec}})
+ m_{\text{transm}} L(RT(x, -\omega_{\text{transm}}), \omega_{\text{transm}})
+ m_{\text{diff}}(x) \int L(RT(x, -\omega), \omega) \cos \theta d\omega
\]

These terms are further simplified in ray tracing.
Global illumination [Kajiya 1986]

\[ L(x, \omega) = \Re( L_{\text{direct}}(x, \omega'), L_{\text{reflected}}(x, \omega'), L_{\text{refracted}}(x, \omega'), L_{\text{ambient}}(x)) \]

\[ = m_{\text{diff}} \sum L_e(\text{RT}(x, -\omega_{\text{light}}), \omega_{\text{light}}) \]

\[ + m_{\text{spec}} L(\text{RT}(x, -\omega_{\text{spec}}), \omega_{\text{spec}}) \]

\[ + m_{\text{transm}} L(\text{RT}(x, -\omega_{\text{transm}}), \omega_{\text{transm}}) \]

\[ + \text{const}_{\text{ambient}} \]

These terms are further simplified in ray tracing.
Introduction to ray tracing

How ray tracing works [Whitted, 1979]

We start with a 3D scene and a camera. What does the camera see?
The blue objects below are specularly reflective but not transparent (opaque mirror-like).
The grey object below is both transparent and specularly reflective (glass-like).

We send a ray from the eye through each pixel on the virtual screen to compute the colour of the pixel. Where does the light travelling along this ray to the eye come from?
OK, now we know where from. Let’s see whether this point is directly illuminated. This one is.
But some light could also have been reflected or refracted by this object. Where does this light come from?
Are those new points directly illuminated? Not both are, only one is. The other light ray is blocked by another object.
What about multiple reflection or refraction? We trace (recursively) the reflected and refracted rays until they do not hit any object in our scene.
Finally we add energy contributions of all rays to get the colour of the screen pixel.
Complexity of ray tracing: number of rays, number of intersections tests.

Estimation: \( \#Rays = \#Eye\_Rays \times N^{D-1} / (N - 1) \times (1 + \#Lights) \)

- \( D \) is the recursion depth
- \( N \) is the average number of rays spawned by a ray-object intersection

Example: \( \#Rays = 640 \times 480 \times 1.2^5 / (1.2 - 1) \times (1 + 20) \approx 80,260,000 \)
Sequential speedup techniques

The most of the ray tracing time (90%) is spent in computing ray-object intersections.

(Sequential) speedup techniques

• Faster ray-object intersections
  • intersection code optimisations
  • bounding volumes
    • bounding volume hierarchies (building BSP trees or octrees from the bounding volumes in the pre-processing phase)
• Tracing fewer rays
  • adaptive recursion depth control
  • vista buffer (saving eye rays by projecting bounding volumes onto the virtual screen in the pre-processing phase)
    • light buffers (saving illumination rays by projecting objects onto a cube around a light source in the pre-processing phase)
• Exploitation of ray coherence
• Tracing more rays at once (tracing cones or cylinders, etc.)
Parallel ray tracing, previous work

**Screen subdivision techniques**

Idea: eye rays are independent, therefore can be computed in parallel without a need of communication between processors

+ easy to implement, sequential speedup techniques can be directly applied

- load balancing, replication of the scene, antialiasing

**Object space subdivision techniques**

Idea: geometrical distribution of objects onto processors, rays or objects are exchanged between processors

+ no scene replication

- difficult to implement, problems with using the sequential speedup techniques, load balancing

**Functional decomposition techniques**

Idea: decomposition of the ray tracing algorithm into subtasks like “ray-object intersection”, “shading” etc. which can be implemented in the client-server way

+ modularity, combination with the previous techniques

- difficult to implement, problems with using the sequential speedup techniques, load balancing
Parallel ray tracing, previous work

Functional decomposition techniques

[Scherson, Caspary, 1988]
- Hierarchical bounding volumes
- Storing the upper part of the tree in every processor; distributing lower parts.
- 2 kinds of tasks on every processor: 1. intersection of a ray with the upper part of the tree; 2. intersection of a ray with the lower part of the tree.
- Idea: the first task will not be running on a processor if workload on the processor is heavy.
- When a ray is to be processed, any processor can take care about the ray. It performs the task 1, then sends a task 2 to the appropriate processor.
- Problem: choosing the level dividing the tree into the upper and lower parts.

[Lefer, 1994]
- A similar approach with more task types
- Implementation on Meiko transputer machine with 8 processors
- Speedup 5.5 - 7 with 8 processors
Parallel ray tracing, previous work

Object space subdivision techniques

[Dippé, Svensen, 1984]

• An algorithm and a parallel computer architecture (3D grid) which fit well together.

• Idea of the algorithm: subdividing the 3D space geometrically into 3D sub-regions (parallel pipeoids, general “cubes”, tetrahedra); mapping 1 sub-region onto 1 processor; exchanging rays between processors.

• Motivation: a mathematical model showing an expected speedup of $O(S^{2/3})$ over the “classical” algorithm ($S$ being the number of sub-regions; the complexity measure was number of ray-object intersections). A sequential testing of all objects against the ray is understood under the “classical” algorithm. No object hierarchy is considered.

• Suggestion for tackling the problem of work imbalance: moving sub-region boundaries.

• Many non-trivial problems, no implementation.

[Cleary, Wywill, Birtwistle, Vatti, 1986]

• The idea of parallelization similar to [Dippé, Svensen, 1984].

• Extended theoretical analysis: $T = T_R + T_P$, where is $T_R$ the time taken to compute the longest ray and $T_P$ is the time taken by the busiest processor; exact analysis for an empty scene.

• No implementation, no object hierarchy is considered.
Parallel ray tracing, previous work

Object space subdivision techniques (continued)

[Priol, Boutatouch, 1988]
• A similar algorithm as in previous papers. Improvement: using pyramidal shapes for regions (by doing so it is guaranteed that the primary rays never leave their initial regions).
• Implementation on an iPSC Hypercube, however, no measurements reported.

[Kobayashi, Nakamura, Shigei, 1988]
• An efficient mapping of 3D sub-regions onto the hypercube. The distance on the hypercube is proportional to the size of the sub-regions.

[Jevans, 1989]
• Idea: an optimistic firing of rays into neighbouring voxel processors before actually computing intersections in the current voxel (and cancelling the rays which should not have been fired).

[Pitot, 1993]
• Implementation of Cleary’s algorithm and experiments on a transputer machine.
• Static load distribution (meta-voxels mapped onto processors, subdividing meta-voxels into voxels on all processors).
Parallel ray tracing, previous work

Object space subdivision techniques (continued)

[Badouel, Boutatouch, Priol, 1994]

- Experiments with *data-oriented parallelization* (object space subdivision):
  - The algorithm depends on empirical choices (size of the sampling grid, etc.)
  - Several regions may share the same object (repeated ray-object computation occur).
  - Processors with light sources are overloaded.

- Experiments with control-oriented parallelization (image space subdivision):
  - Work stealing in a logical processor ring; object cache (inspired by Paddon and Green).
Parallel ray tracing, previous work

Object space subdivision techniques (continued)

[Keates, Hubbold, 1995]

• Experiments with Kendal Square Research Machine (KSR1)

• Distributed memory, virtual shared memory supported by hardware, pthread programming model.

• A very good characteristics: efficiency 75% on 200 processors, 70% on 230 processors.
Parallel ray tracing, previous work

**Screen subdivision techniques**

[Green, Paddon, 1989]

A virtual memory management system (on every worker processor)

- Resident object set (always in local memory)
- Object cache (used during the computation for storing non-resident objects in order to minimize the communication)
- Additional processors with enough memory acting as object database server(s)

The resident object sets are determined by tracing a very low resolution picture in the pre-processing phase.

Implementation: a simple ray tracer (4-vertex polygons, octree object hierarchy), up to 10 T800 Transputers, LRU cache, static load balancing.

```java
if (cached object needed) {
    if (the object is in permanent database or in the local cache)
        return(object)
    else {
        send request to the processor owning the object()
        wait for the object()
        return(object)
    }
}
```
POV||Ray: Persistence Of Vision Parallel Ray Tracer

(Plachetka, 2003)

Screen subdivision
Dynamic load balancing

Data parallelism

- Caching *mesh objects* only; bounding box hierarchy is kept intact, only data (vertices, normals, etc.) are distributed.

- Checking for the local presence of the data and eventual fetching the missing data from another processor occurs only in one place in the source code (intersection calculations).

- Pre-computing information needed later during the lighting calculations and associating it with the intersection point (normal, mapped bitmap colour, etc).

- A call-back mechanism is used for answering OBJECT_REQ messages. These messages can be processed directly in an application-independent thread.

- LRU caching strategy was selected as the best one (previous experiments also with other strategies, LRU_COUNTER, RANDOM, AGING).

Tomas Plachetka, Comenius University, Bratislava
Bratislava, 30 Mar 2006 15
Motivations for a data-parallel implementation:

- No limit on the maximum scene size.
- By large scenes, in order to avoid swapping on processors, the total local memory should be limited.

Solution: central memory management, dynamic (run-time) cache size adjustment.
POV||Ray: Persistence Of Vision Parallel Ray Tracer

hpcLine: HAUS6 (Arcon), 614 objects (72,621 triangles), 22 point lights (640x480)

<table>
<thead>
<tr>
<th>Processors</th>
<th>Linear speedup</th>
<th>Speedup, no AA</th>
<th>Efficiency, no AA</th>
<th>Speedup, AA 4x4</th>
<th>Efficiency, AA 4x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1,00</td>
<td>1,00</td>
<td>1,00</td>
<td>1,00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2,00</td>
<td>1,00</td>
<td>2,03</td>
<td>1,02</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4,00</td>
<td>1,00</td>
<td>4,08</td>
<td>1,02</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>7,97</td>
<td>1,00</td>
<td>8,03</td>
<td>1,00</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>15,79</td>
<td>0,99</td>
<td>15,47</td>
<td>0,97</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>22,66</td>
<td>0,94</td>
<td>22,28</td>
<td>0,93</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>29,54</td>
<td>0,92</td>
<td>28,99</td>
<td>0,91</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>34,26</td>
<td>0,86</td>
<td>34,34</td>
<td>0,86</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
<td>41,19</td>
<td>0,86</td>
<td>38,03</td>
<td>0,79</td>
</tr>
</tbody>
</table>

![Graph: HAUS6 (100% memory, warm start)](chart1.png)

![Graph: HAUS6 (100% memory, warm start)](chart2.png)
hpcLine: HAUS6 (Arcon), 614 objects (72,621 triangles), 22 point lights (640x480)

- Total object requests: ~50,000,000
- Cold start, 48 workers, 100% memory: 3392 objects exchanged (~13 MB)

<table>
<thead>
<tr>
<th>Processors</th>
<th>Objects exchanged</th>
<th>Bytes exchanged</th>
<th>Exchange time (s)</th>
<th>Cache hit ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>24</td>
<td>713.562</td>
<td>inf</td>
<td>2.104.2</td>
<td>0.9860</td>
</tr>
<tr>
<td>32</td>
<td>501.686</td>
<td>2.111.045.688</td>
<td>1.926.0</td>
<td>0.9902</td>
</tr>
<tr>
<td>40</td>
<td>426.791</td>
<td>1.798.978.892</td>
<td>1.819.7</td>
<td>0.9917</td>
</tr>
<tr>
<td>48</td>
<td>357.778</td>
<td>1.517.580.352</td>
<td>1.466.9</td>
<td>0.9930</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processors</th>
<th>Objects exchanged</th>
<th>Bytes exchanged</th>
<th>Exchange time (s)</th>
<th>Cache hit ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>24</td>
<td>9.764.151</td>
<td>inf</td>
<td>25.200.7</td>
<td>0.7671</td>
</tr>
<tr>
<td>32</td>
<td>7.244.288</td>
<td>inf</td>
<td>24.465.3</td>
<td>0.8357</td>
</tr>
<tr>
<td>40</td>
<td>5.539.805</td>
<td>inf</td>
<td>24.880.1</td>
<td>0.8799</td>
</tr>
<tr>
<td>48</td>
<td>5.054.140</td>
<td>inf</td>
<td>25.732.2</td>
<td>0.8921</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processors</th>
<th>Objects exchanged</th>
<th>Bytes exchanged</th>
<th>Exchange time (s)</th>
<th>Cache hit ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>24</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>32</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>40</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>48</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processors</th>
<th>Objects exchanged</th>
<th>Bytes exchanged</th>
<th>Exchange time (s)</th>
<th>Cache hit ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>24</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>32</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>40</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>48</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
POV||Ray: Persistence Of Vision Parallel Ray Tracer

• Screen parallelisation can work with a distributed object database.
  • Almost all sequential parallelisation techniques can be reused!
  • Object distribution can be treated as an independent problem

• A good screen parallelisation (process farming) algorithm and its tuning can be found in [Plachetka 2004, Tuning of Algorithms…],