

Partitioning Screen Space 2



Rui Wang

- Architectural Implications of Hardware-Accelerated Bucket Rendering on the PC (97')
- Dynamic Load Balancing for Parallel Polygon Rendering (94')
- Models of the Impact of Overlap in Bucket Rendering (98')

Architectural Implications of Hardware-Accelerated Bucket Rendering on the PC

- Bucket Rendering
 - Divide scene into screen-space tiles and render each tile independently
 - Advantages
 - Smaller memory requirement
 - Parallel rendering
 - Disadvantages
 - Overlap of primitives over multiple buckets
 - Overhead of sorting primitives into buckets

Bucket Rendering

- RenderMan, PixelPlanes 5, PixelFlow, Apple systems, Talisman
- Choosing tile size
 - Smaller memory requirement, better load balancing, but more overlaps
 - 32x32 tile with dense SRAM
 - 128x128 tile with embedded DRAM

Architecture

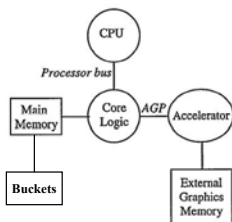
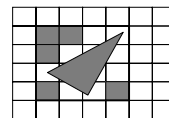


Figure 1. PC Architecture.

Bucket Sorting

- Screen-aligned bounding box

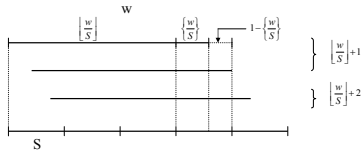


- Exact bucket sorting

Analytical computation of overlap

- Molnar 91', modified to consider r

■ Equation: $E_o = (1 + \frac{w}{S}) \times (1 + \frac{h}{S})$



More Equations ...

$$\left[\left(\frac{w}{S} + 1 \right) \times \left(\frac{h}{S} + 1 \right) \right] \times \left[\left(1 - \frac{w}{S} \right) \times \left(1 - \frac{h}{S} \right) \right] + \text{blah, blah, blah}$$

$$E_o = \left(1 + \frac{w}{S} + \frac{w}{S} \right) \times \left(1 + \frac{h}{S} + \frac{h}{S} \right) = \left(1 + \frac{w}{S} \right) \times \left(1 + \frac{h}{S} \right)$$

⇓

$$r = \frac{w}{h}, s^2 = w \times h \Rightarrow E_{or} = 1 + \left(\sqrt{r} + \frac{1}{\sqrt{r}} \right) \times \frac{s}{S} + \left(\frac{s}{S} \right)^2$$

⇓

minimized when $r=1$ $E_o = \left(1 + \frac{s}{S} \right)^2$ or $O = \left(\frac{S + \sqrt{\rho a}}{S} \right)^2$

- Overhead incurred by overlap

- Main memory bandwidth
- AGP bandwidth
- Primitives set-up

- Experiments

- Intel Scene Management
- Trace polygons rendered
- Bucket size 16, 32, 64, 128, 256

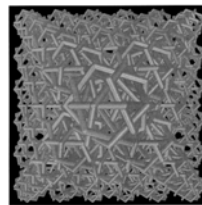
Results

- Larger size tri doesn't matter. Many primitives that are smaller than 1K pixel size have with large aspect ratio (Fig.2-9)
- Expected E_o assuming $r=1$ closely fit observed overlap => Molnar equation is still a good prediction assuming $r=1$ (Tab.2)
- Bounding box area scales with screen resolution

Discussion


- Assume triangle set-up: 1M/s
Bandwidth of memory: 200~300MB/s
AGP bandwidth: 256MB/s
- 128x128 tile: resource consumption stay below 25%, 15%, 10% respectively for all resolutions (Tab.3)
- 32x32 tile: resource consumption prohibitively high above 1024x768 (Tab.4)

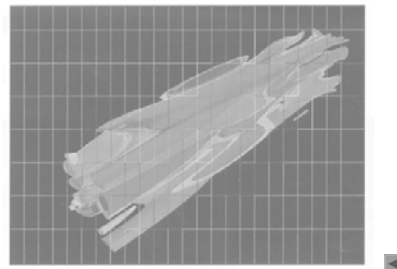
Dynamic Load Balancing for Parallel Polygon Rendering



A New graphics renderer incorporates novel partitioning methods for efficient Execution on a parallel computer. It requires little overhead, executes efficiently, and demands minimal processor synchronization.

Image-space Parallel Algorithms

- Horizontal, vertical strips, rectangular areas
- Load balancing:
 - Data nonadaptive: statically, dynamically
 - Data adaptive: less suitable for a single image
- Data nonadaptive
 - Tile image space to $R \times P$ rectangular areas 
 - Tradeoff to choosing R (20 is good)
 - This paper uses $2 \times P$ by applying a *task adaptive* work decomposition strategy



Algorithm Description

- Preprocessing: 13%
 - Data read-in, xforming, normal calc., back-face culling, clipping, perspective proj.
- Rendering: 86%
 - Hidden-face removal, shading, anti-aliasing, etc
- Post processing: 1%
 - Display frame buffer or store in file
- Fig.2

Preprocessing

- Number of objects $> P$
 - Round-robin distribution
- Number of objects $< P$
 - Split into multiple sub objects
 - Parallel xforming or clipping; Reader process assigns data to individual processors
- Sort
- Implementation on BBN TC2000 (96 processors), speedup 9.4

Rendering

- Modified scan-line z-buffer algorithm with stochastic sampling (16 per pixel)
- $R=2$: preprocessing overhead reduced
 - The first P regions assigned to P processors
 - Dynamically get from the additional P tasks
 - **partition:** (steal tasks from other processors)
 - Search for the processor with most work left P_{max}
 - If there is sufficient work to do, proceed, otherwise return
 - Set locks to prevent concurrent stealing
 - Partition P_{max} work into two segments, take the lower one
 - Unset locks and work on the new task

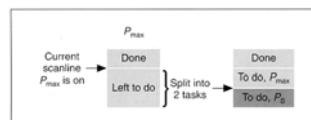


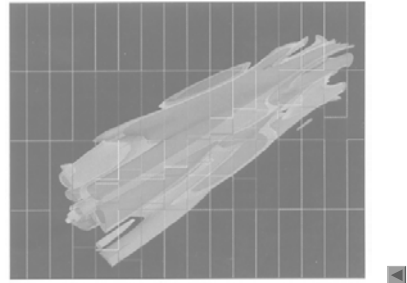
Figure 4. The splitting process.

- The splitting strategy maintains coherence at P_{max} side without interruption
- With task adaptive approach, load balancing is good even for $R=2$
- Effect of initial region aspect ratio on splitting: 2:1 worst, 1:3 best

Some Issues

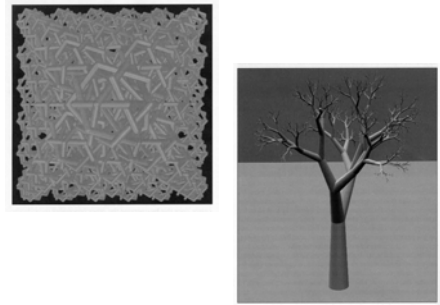
- Find the remaining work on a processor
 - Record and update the current remaining scan lines
 - Fig.5
- Avoid race conditions or deadlocks
 - Test and lock: if not locked, store in `p_max`; if locked, store in `pot_p_max`
- After getting through a lock, the work might have been completed meanwhile

Post Processing ...



Data Distribution

- Locally Cached (LC)
 - Distributing data among memories
 - Copy exactly the amount of data necessary for computation to local memory
 - After preprocessing, each processor queries and retrieves data for rendering from other processors
 - Data movement during partitioning
 - Quick clip after data retrieved, instead of obtain exactly the lower region task => reduce the amount of data for further partitioning



- Load balancing overhead increases with P
- Communication Overhead also increases with P

Figure 9: Measured degradation factor as a function of P for the scene data set.

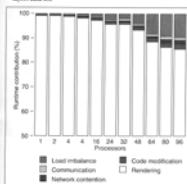


Figure 10: Measured degradation factor as a function of P for the ship data set.

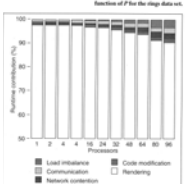
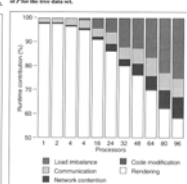


Figure 11: Measured degradation factor as a function of P for the tree data set.



Models of the Impact of Overlap in Bucket Rendering

- Bucket Rendering
 - Subdivide frame buffer into coherent regions that are rendered independently
 - Tile size is a design decision
 - Advantages and disadvantages

Analytical Models

- Fig.1 defines key terms
- Per-pixel cost=1
- Per-triangle cost for non-tiled=k,
- Tiled=kO
knocout...

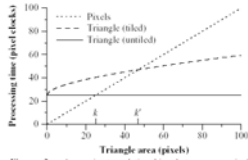


Figure 2: Approximate relationships between per-pixel computation, per-triangle computation in an untiled system, and per-triangle computation in a tiled system (the original per-triangle computation multiplied by the overlap factor). Assumes k set to 25 and S set to 32.

Modeling Overlap

- Overlap factor: $O = \left(\frac{S + \sqrt{\rho a}}{S} \right)^2$
- Source of error:
 - Assume unit aspect ratio, best for overlap
 - Assume uniform tri area, worst for overlap
 - Assume $\rho = 3$

Software Model

- Assume single processing unit for per-triangle and per-pixel work

$$R_{sw} = \frac{kO + a}{k + a}$$

$$\frac{dR_{sw}}{da} = 0$$

$$\lim_{a \rightarrow \infty} R_{sw} = 1 + \frac{k\rho}{S^2}$$

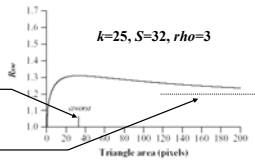
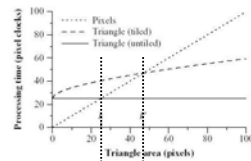


Figure 4: The graph above shows the expected ratio of rasterization time with tiling to that without tiling for a software system (R_{sw}) as triangle area (a) changes. The values for the graph are computed from Equation 2 with k set to 25, ρ set to 3, and S set to 32.

Hardware Model

- Assume perfect pipelining (infinite buffer); processing time depends on the slowest unit
- Processing time on real systems (Fig.3)



$k' = kO = a$

Hardware Model

- Non-tiled: Tiled:

$$\max(k, a) = \begin{cases} k, & 0 < a \leq k \\ a, & k < a \end{cases} \quad \max(kO, a) = \begin{cases} kO, & 0 < a \leq k' \\ a, & k' < a \end{cases}$$

$$R_{hw} = \frac{\max(kO, a)}{\max(k, a)} = \begin{cases} O, & 0 < a \leq k \\ \frac{kO}{a}, & k < a \leq k' \\ 1, & k' < a \end{cases}$$

$a_{worst} \approx 25$

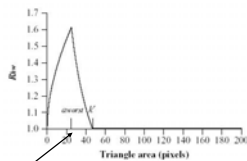


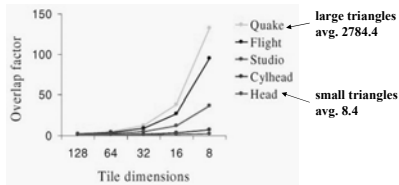
Figure 5: The graph above shows the expected ratio of rasterization time with tiling to that without tiling for a hardware system (R_{hw}) as triangle area (a) changes. The values for the graph are computed from Equation 3 with k set to 25, ρ set to 3, and S set to 32.

Observations of HW Model

- Inefficiency of tiled rendering is limited
- The worst case occurs at exactly the triangle area where processing the triangle requires as much time as processing its pixels ($a_{worst} = k$) => pipeline balanced

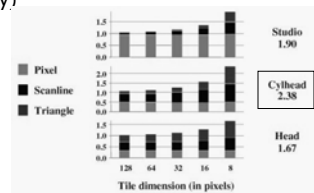
Experiments

- How well the mathematical model works
- Datasets



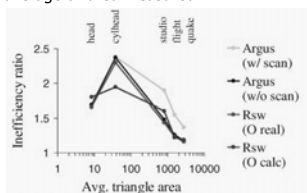
Experiments on SW Model

- OpenGL Stream Codec (GLS)
- Argus rendering system on top of SimOS (full machine simulator, measurement taken at pixel granularity)



Experiments on SW Model

- Decide k , O , a :
 - $k=9.3$ (non-textured), $k=3.6$ (textured)
 - O_{real} is measured, O_{calc} is predicted
 - a =average tri area measured

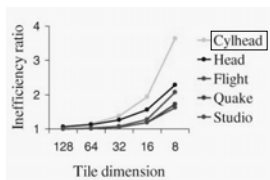


Analysis on SW Model

- Worst efficiency occurs at moderate size (conforms to the model)
- Limitation:
 - Error of overlap estimation using Molnar equ.
 - Does not consider scan line processing time

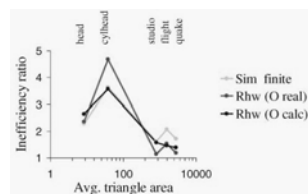
Experiments on HW Model

- Single triangle buffer vs. Infinite



Experiments on HW Model

- Decide k , O , a :
 - $k=25$
 - O_{real} is measured, O_{calc} is predicted
 - a =average tri area measured



Analysis on HW Model

- The model correctly predict inefficiency peaks at triangle area close to $k=25$
- Interesting thing to understand (???)
 - More buffer space is most effective when average tri area match k (pipeline balanced)
 - For large avg. triangles, tiling moves effective tri area closer to k , thus infinite buffer takes effect
 - For small avg. triangles, tiling moves effective tri area below k , thus infinite buffer has little effect
- Overlap calc error still exists

Conclusion

- The impact of overlap is highest when work is balanced between triangle processing and pixel processing
- The impact of overlap in pipelined system is limited to a window of tri size around the design point of the system
- Has limitations