Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip*

서강대학교 컴퓨터 공학과 그래픽스 연구실
오상락
2008-07-22

* [SCHMITTLER et al. 2004]

Overview

- SaarCOR Hardware Architecture [SCHMITTLER et al. 2002]
  - Flexible, modular, and scalable hardware architecture for real-time ray tracing.
  - 시뮬레이션 결과 OpenGL-like scene에서 유사한 사양의 Rasterization Architecture와 비슷한 성능을 보임을 밝혔음.

- Ray Tracing of Dynamic Scene
  - 오브젝트 단위의 동적인 아핀 변환을 효율적으로 지원함.
  - 아핀 변환을 위한 기하변환 유닛은 다른 계산에도 효율적으로 사용될 수 있음.

- FPGA Chip
  - VHDL이나 베릴로그와 같은 하드웨어 기술 언어로 프로그래밍 할 수 있는 칩.
  - 개발시간이 짧고, 오류수정을 현장에서 재 프로그램할 수 있고, 초기 개발비가 저렴한 장점이 있음.
  - 일반적으로 주문형 반도체(ASIC)보다 느리고 복잡한 설계에 적용할 수 없으며 소비전력이 큼.

Ray Tracing of Dynamic Scenes

- Ray Tracing은 씬의 크기에 대해 Spatial index(Grid, kd-tree, etc.,)를 통해 로그 함수 정도의 복잡도를 가지는 장점이 있음.
  - Rasterization 방법은 선형 복잡도를 가짐.

- Spatial index 생성은 최소한 선형 복잡도를 가지기 때문에 동적인 씬에서의 ray tracing의 장점이 없어짐.
  - 일반적인 씬에서 많은 부분이 정적이기 때문에 모든 인덱스를 매번 구성할 필요는 없음.

- 동적으로 바뀐 오브젝트의 Spatial index를 새로 만들지 않고, 이전 Spatial Index를 아핀 변환한 뒤 해당 위치의 탑 레벨 Spatial index에 삽입함.
  - Spatial index를 다시 처음부터 제작하는 것보다 품질이 안 좋음.
  - 동적인 변환이 어느 정도 제한된 환경에서는 많은 효과가 있음.

Ray Tracing of Dynamic Scenes (cont'd.)

- 장면 오브젝트 각각에 대해 아핀 변환 행렬을 가짐.
  - 오브젝트들의 interactive한 아핀 변환 지원.
  - Multiple instance의 효과적인 지원.

- Ray가 탑 레벨 인덱스와 히트 하면 해당 오브젝트의 좌표계로 변환되어 인덱스 탐색을 계속하게 됨.

- 해당 방식은 Ray의 잦은 아핀 변환이 요구되므로 많은 하드웨어 리소스를 필요로 하게 되고, 기하 변환 유닛이 필요해짐.
  - 기하 변환 유닛은 많은 floating point연산이 필요하기 때문에 하드웨어 코스트가 비싼 유닛임.
    - ray의 아핀 변환에만 사용된다면 대부분의 시간에 idle상태에 있는 문제가 있음.
  - 하드웨어의 기하 변환 유닛을 다른 계산에도 사용해 전반적으로 68%의 코스트를 줄였음.
    - Shading, Secondary Ray generation 등.

Ray Triangle Intersection

- Ray Triangle Intersection은 두 스테이지로 구성됨.
  1. Ray가 해당 트라이앵글의 아핀 변환을 따라 변환됨.
     - 변환된 좌표계의 삼각형은 단위 삼각형임.
  2. 단위 삼각형에서의 Intersection 검사는 매우 간단하게 이뤄질 수 있음.

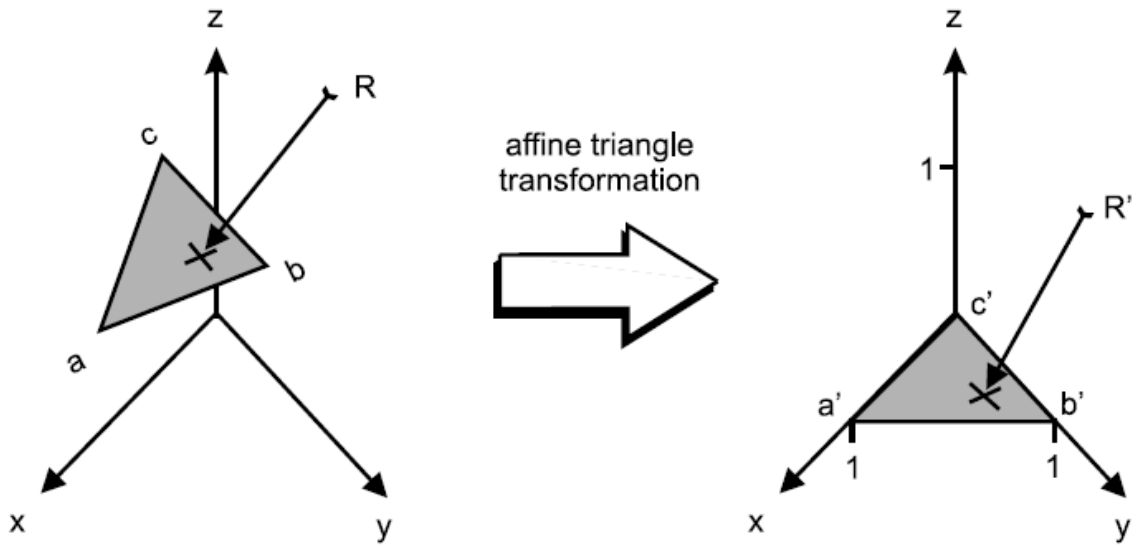
- Affine Triangle Transformation
  - 해당 삼각형을 단위 삼각형 space로 변환시키는 아핀 변환을 구한다.

- Unit Triangle Intersection
  - 단위 삼각형 space상에서 Intersection 검사를 하게 된다.

- Dot Product Preservation
  - 단위 삼각형 space상에서, Ray 방향 벡터와 삼각형의 Normal의 Dot Product는 특별히 계산할 필요 없이 바로 구할 수 있다.

Ray Triangle Intersection (cont'd.)

- Affine Triangle Transformation
  - 삼각형 $\Delta = (A,B,C)$ with $A,B,C \in \Re^3$ 의 아핀 변환은 노말이 $N_{unit} = (0,0,1)$ 인 단위 삼각형으로 바꿔주는 변환 $T_\Delta(X) = m \cdot X + N$ with $m \in \Re^{3,3}$ and $X,N \in \Re^3$.



(a) world coordinate space (b) unit triangle space

Ray Triangle Intersection (cont'd.)

- Affine Triangle Transformation (cont'd.)
  - 변환 $T_{\Delta}^{-1}(X)$은 다음 방정식을 통해 쉽게 구할 수 있음.

$$T_{\Delta}^{-1}\begin{pmatrix}1\\0\\0\end{pmatrix}=A \quad T_{\Delta}^{-1}\begin{pmatrix}0\\1\\0\end{pmatrix}=B$$

$$T_{\Delta}^{-1}\begin{pmatrix}0\\0\\0\end{pmatrix}=C \quad T_{\Delta}^{-1}\begin{pmatrix}0\\0\\1\end{pmatrix}=N$$

  - 구해진 변환 $T_{\Delta}^{-1}(X)=\begin{pmatrix}A_x-C_x & B_x-C_x & N_x-C_x\\ A_y-C_y & B_y-C_y & N_y-C_y\\ A_z-C_z & B_z-C_z & N_z-C_z\end{pmatrix}\cdot X+\begin{pmatrix}C_x\\C_y\\C_z\end{pmatrix}.$

  - $T_{\Delta}^{-1}(X)$는 잘 정의되며 유일하다. 만약 삼각형이 퇴화 삼각형(degenerate triangle)이 아니라면 역 변환 $T_{\Delta}(X)$이 존재하며, 이는 one-to-one 아핀 변환이다.

Ray Triangle Intersection (cont'd.)

- Unit Triangle Intersection
  - Intersection 결과는 one-to-one 아핀 변환 아래에서도 동일한 t값과 barycentric coordinate를 가짐.
    - world coordinate space에서와 단위 삼각형 space에서의 intersection결과는 같음.
  - 삼각형이 unit triangle이기 때문에 다음과 같이 간단하게 구할 수 있다.
    - 단위 삼각형 space 상의 Ray R'을 구한다.

$$\text{Let } R' = T_\Delta(R) =$$
$$T_\Delta(O,D) = (m \cdot O + N, m \cdot D) = (O', D')$$

    - Intersection은 다음과 같이 계산 될 수 있음.

$$t = -\frac{O'_z}{D'_z},\ u = O'_x + t \cdot D'_x,\ \text{and}\ v = O'_y + t \cdot D'_y.$$

Ray Triangle Intersection (cont'd.)

- Dot Product Preservation

$$D' \cdot N_{unit} = D'_z.$$

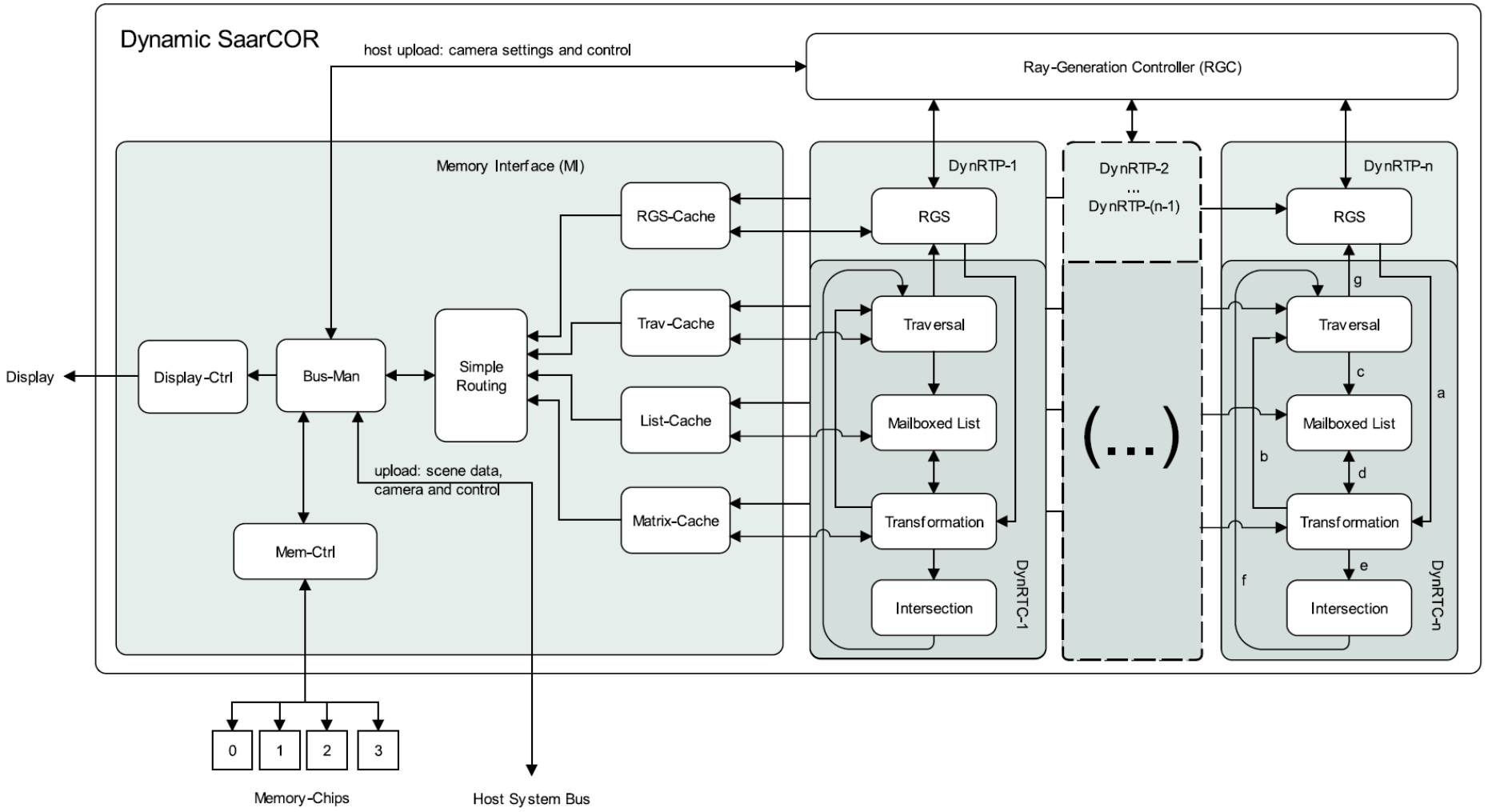
Shading

- Shading 계산을 할 때 기하 변환 유닛 필요.
  - Object Space에서 World Space로 Normal등을 변환 해야 됨.

- Ray Generation(Primary, Shadow, Reflection, Transparency, Refraction)도 기하 변환 유닛 사용.
  - 기술적인 디테일은 [SCHMITTLER et al. 2004] 을 참조.

- 기하 변환 유닛이 Dynamic Raytracing뿐만 아니라 Shading에서도 쓰이기 때문에 해당 유닛의 이용률을 높일 수 있다.

Review: SaarCOR Hardware Architecture



Prototype and Implementation

- 프로토 타입은 FPGA 하드웨어에서 6개월 동안 3명이 만들었음.
  - FPGA칩: Xilinx Virtex-II 6000-4 FPGA
  - 호스트 보드: Alpha Data ADM-XRC-II PCI-board
    - 6 independent banks of 32-bit wide SRAM (each 4MB) running at the FPGA clock speed
    - PCI-bridge
    - general purpose I/O-channel. This channel is connected to a simple digital to analog converter implementing a standard VGA output supporting resolutions of up to 1024x768 at 60Hz.
  - 개발 도구: JHDL, Xilinx tool.
  - OS: Linux

Prototype and Implementation (cont'd.)

- 벤치 마킹 결과, floating-point 연산은 24bit floating-point number, 16bit mantissa가 하드웨어 비용과 정확도 측면에서 가장 좋았다고 함.
  - 일반적인 그래픽 하드웨어와 같음.

- Shading을 위한 메모리 대역폭은 낮아도 되고, chache도 필요 없음.
  - Shading 데이터는 Ray당 28바이트 만 있으면 됨.
  - 결과 픽셀 색만 frame buffer에 쓰면 됨.

- 프로토 타입에선 FPGA칩의 한계로 하나의 intersection 유닛만 구현했음.
  - [SCHMITTLER et al. 2002]가 일반적인 씬에서 traversal이 intersection보다 4배 정도 더 수행됨을 보였음.
  - 4개의 ray로 이뤄진 packet을 병렬로 traverse하고, intersection test는 순차적으로 함.
  - 만약 유닛을 필요한 만큼 다 쓰게 된다면 90 MHz에서 2GB/s 정도의 대역폭 필요.
    - 캐쉬를 쓰면 1GB/s정도로 떨어질 수 있음.

Prototype and Implementation (cont'd.)

- 한 개의 패킷당 64개의 kd-tree traversal 상태를 저장할 수 있는 스택 제공.

- 필요한 메모리와 계산량이 매우 적음.(Table 2 참조)
  - 64개 쓰레드와 256개의 라이팅 제공, VGA 포트 제공
  - 프로토 타입에서 FPGA 로직 셀의 56% 밖에 못 썼고, 78% 의 메모리만 썼음.
  - 90MHz에서 40억 FLOPs로 구현되었음.
    - GeForce 5900FX는 500MHz에서 2000억 FLOPs. 약 50배.

Unit	Add	Mul	Div	Cmp	Mem
Traversal	4	–	4	13	44.5 KB
Mailboxed List	–	–	–	–	0.8 KB
Transformation	9	9	–	–	9.3 KB
Intersection	3	2	1	–	–
DynRTC-Cache	–	–	–	–	15.6 KB
Shader	–	–	–	–	4.8 KB
Total	16	11	5	13	75.0 KB

Table 2: *Complexity of one ray tracing pipeline measured in floating-point units for addition, multiplication, division, and comparison, respectively. The rightmost column also lists the internal memory requirements including any meta-data and global state, such as parameters for 8 light-sources. Each DynRTP uses 32 threads and contains caches that store 512 data-items each.*

Prototype and Implementation (cont'd.)

- 요즘 나오는 FPGA 칩은 더 사양이 좋아서 두 개 이상의 파이프라인을 구현할 수 있음.

- 보통 그래픽 카드와 외부 메모리 대역폭이 30GB/s이므로 대역폭 상으로는 100개 이상의 독립적인 Ray tracing 파이프라인을 넣을 수 있음.

Benchmark-Scenes and Results

- OpenRT와 비교
  - Pentium-4 with 2.66GHz and 1GBRAM.
  - SSE-optimized code on packets of four rays.

- [SCHMITTLER et al. 2002]와 실험 환경 상 2가지 면에서 달라진 게 있음.
  - KD-tree 퀄리티를 개선했음.
  - CPU에선 비용이 많이 드는 textured shading을 했기 때문에 CPU에 약간 불리했을 것 같음.

Benchmark-Scenes and Results (cont'd.)

- CPU Clock이 30배는 빠른데 비해, 하드웨어가 3~5배는 CPU보다 빠른 결과를 보였음.
  - Multi-threading approach로 칩의 사용률이 높았음.
  - Clock당 floating point 연산의 효율성이 높았음.

			#frames per second		
Scene	#triangles	#objects	SaarCOR	OpenRT	Speed-Up
Scene6	806	1	60.8	12.9	4.7
Castle	20 891	8	23.8	9.2	2.6
Office	34 312	1	48.9	10.4	4.7
Quake3	39 424	1	33.6	11.1	3.0
Quake3-p	52 790	17	26.7	7.9	3.4
UT2003	52 479	1	25.4	8.0	3.2
Conference	282 805	54	22.1	8.1	2.7
Terrain	10 469 866	264	15.9	3.5	4.5
SunCOR	187 145 136	5622	32.1	7.5	4.3

Conclusion and Future Work

- Rasterization approach처럼 Ray tracing도 하드웨어에 구현에 적합하다는 것을 보여줬음.

- 간단한 프로토 타입에서도 다양한 씬에 대해 real-time의 퍼포먼스를 보임.

- Rasterization 하드웨어의 두 가지 큰 문제를 해결할 수 있을 것임.
  - 외부 메모리와의 Bandwidth가 비교적 매우 적음.
  - 매우 쉽고 효율적으로 확장 가능함.
    - 칩 안에서 여러 파이프라인.
    - 보드에서 여러 칩.
    - pc에서 여러 보드.

Conclusion and Future Work (cont'd.)

- 확장성은 씬 데이터의 Bandwidth에 제약을 받는데, cacheing이나 read-only 데이터의 복제 등으로 해결 가능.

- Dynamic Scene의 지원이 실제로는 기하 변환 모듈의 다양한 활용 덕분에 고정된 간단한 static 구현보다 25%는 하드웨어 코스트가 적고, 다른 Dynamic 구현에 비해서는 68%나 좋다고 함.

- OpenRT API가 하드웨어 Raytracing 엔진에 적합할 것으로 보임.

- 오브젝트 단위로 아핀 변환하는 방법엔 어느 정도 한계가 있음.
  - 더 많은 유연성이 필요.

- 프로그래머블 쉐이딩을 앞으로 제공할 필요가 있음.

References

[SCHMITTLER et al. 2002] SCHMITTLER, J., WALD, I., AND SLUSALLEK, P. 2002. SaarCOR – A Hardware Architecture for Ray Tracing. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, 27–36.*

[SCHMITTLER et al. 2004] SCHMITTLER, J., WOOP, S., WAGNER, D., PAUL, W. J., , AND SLUSALLEK, P. 2004. Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip. In *Proceedings of Graphics Hardware.*