A presentation on my background in computer graphics

M. Mostajab

www.mmostajab.com

May 9, 2017

Outline

About Me...

- My name is **Morteza Mostajab**
- B.Sc. Computer Engineering at
  Hamedan University of Technology, Iran
- M.Sc. Computer Science at
  Technische Universität München
- Working as Researcher at
  Fraunhofer IGD, Darmstadt
- Research interests:
  Real-time physically-based rendering
  (Rasterization-based or Ray-tracing)
  Virtual reality
  Computer graphics and visualization
  Game Programming

Inspiration

- Games, Animations, Movies with Special Effects,...


(a) Last Ninja 3


(b) Gears of War


(c) Ratatouille


(d) The lord of the rings

- My firsts...

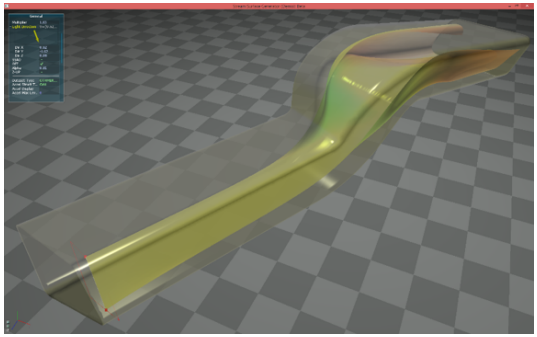


(e) First Computer (Commodore 64)

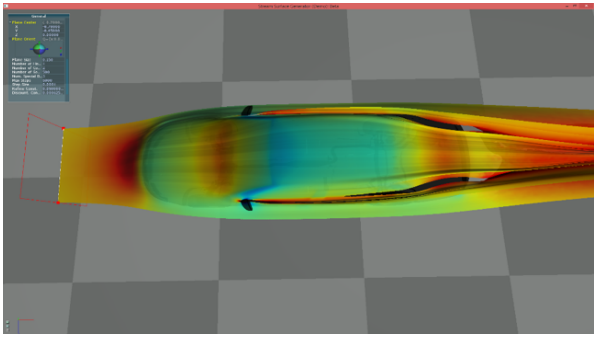

(f) First IBM compatiable PC (80286)


(g) First Console (Atari 2600)

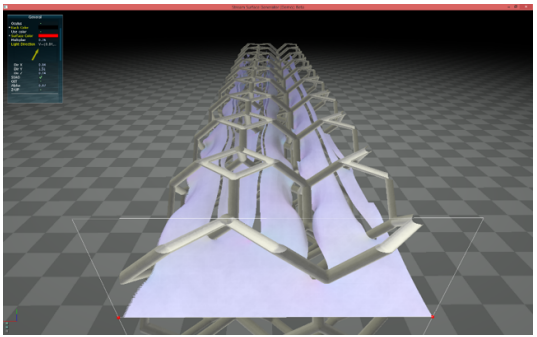
Outline



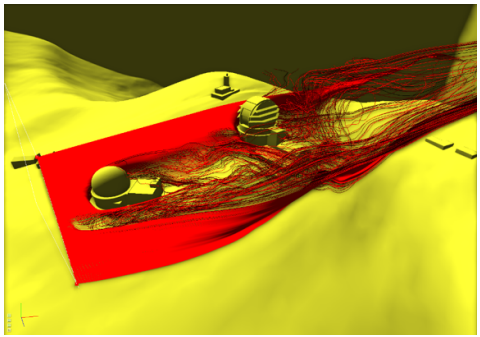
(h) Wind Tunnel



(i) Car



(j) Tethrahedradecal



(k) Telescope

Figure 1: Final results of this master thesis.

Motivation

- **Goal**: Use virtual reality to help engineers getting a better **understanding** of simulations, make **investigation** in flow field features easier.
- Virtual reality applications demands 75-140 frames per second scene update and rendering to reduce **latency**.
- **Initial motivation**:
  – **Immersive Streamline Demo**: a demo which user could stand in a simulation model to understand flow field features by:
  - Prototype demo which placed the user virtually in simulation model.
  - User could interact with virtual world by moving streamlines' seeding plane.
  - User could walk around in the simulation model, look closer into streamlines to understand flow field's features better.

Immersive Streamline Demo

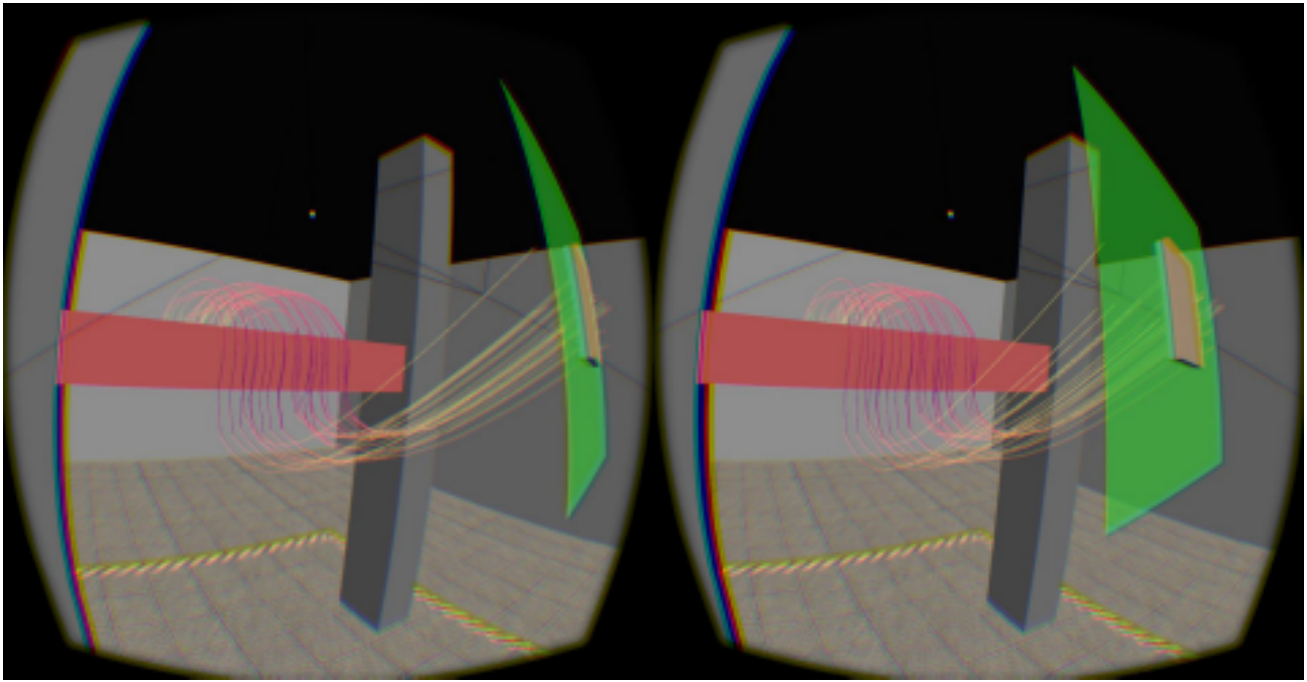


Figure 2: **Immersive Streamlines** was a prototype demo done at Fraunhofer IGD. The user is able to move inside the virtual world, interact with it, and look closely at the flow visualization results. The image shows a stereo pair, which is displayed on Oculus Rift HMD.

Problem Definition

- Streamline's computation was 10 frames per second (latency was very visible to user).
- Streamline's accuracy was not high enough (straight lines in streamlines are visible in the screenshot).
- Stream surfaces can provide more information about flow fields features.
- So, we defined **Parallel Stream Surface Computation and Rendering** master thesis to solve these problems.

Main Contributions

- **Contribution 1**: Using heterogeneous computing:
  – Scale streamline computation on all capable devices.
  – Scale rendering on all graphic processing units.
- **Contribution 2**: Investigation techniques from ray tracing field for application in accurate streamline computation:
  – Using acceleration structures
  – Using ray-packing

Benefits

- Accurate streamline computation method (using **Runge-Kutta** integration method).
- **Adaptive Runge-Kutta** integration method to adapt steps to maximum integration error.
- Heterogeneous stream surface computation.
- Multi GPU stream surface computation and result

Related Works

- Parallel stream surface computation for large data sets (Camp et al., 2012).
  proposed a distributed stream surface computation system.
  – **My method is not distributed but uses all available computation devices.**
- Interactive Streak Surface Visualization on the GPU (Buerger et al., 2009).
  Bue+09 samples the simulation mesh on a regular grid, then compute the streak surface.
  – **Reduces accuracy, ignores a lot of information exist in simulation mesh by sampling it on a regular grid.**

Related Works

- Interactive particle tracing for the exploration of flow fields in virtual environments (Schirski, 2008).
  Sch08 uses the neighboring graph instead of acceleration structure on GPU.
  – **step size needed to be small enough.**
- Fast, Memory-Efficient Cell Location in Unstructured Grids for Visualization (Garth and Joy, 2010).
  introduces cell-tree acceleration structure use it for particle tracing.
  – **I have evaluated more acceleration structures to classify them based on memory requirements and performance.**

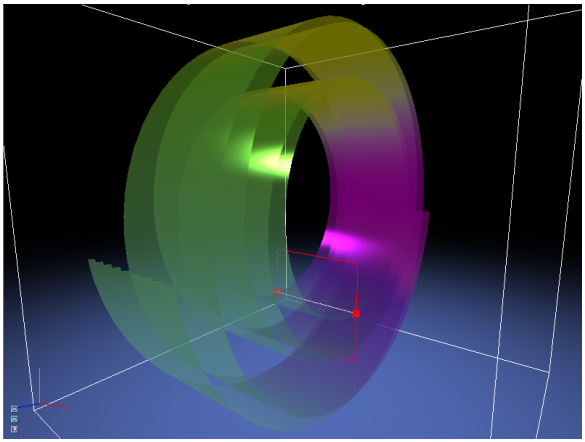
Stream Surface Computation Algorithm (Camp et al., 2012)

Data: *seeds* list which contains all initial seeding points.
Result: Stream Surface Points
while *seeds list is not empty* **do**
  Tracing: Trace streamlines originating from seeding points stored in *seeds* list and make *seeds* empty.
  Refinement:**for** *each pair of adjacent streamlines* s_1 *and* s_2 **do**
    $d \leftarrow Distance(s_1, s_2)$
    if $d \leq D_{disc}$ **then**
      Discontinuity: The surface is discontinued in that area.
    else
      $s_{new} \leftarrow (s_1[0] + s_2[0])/2$
      add s_{new} to list *seeds*.
    end
  end
end

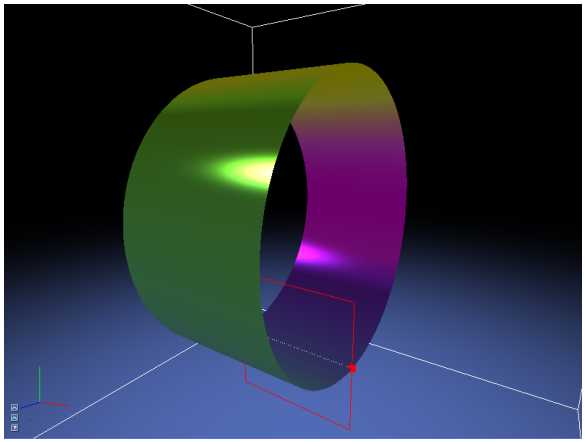
Algorithm 1: Stream Surface Computation Algorithm (Camp et al.,

Accurate Streamline Computation

- Point-Inside-Cell checks & interpolating results inside cells.
- Using *adaptive Runge-Kutta* integration method (Cash-Karp).
- Adaptive stepping (faster and more accurate integration).
- Avoiding early terminations.



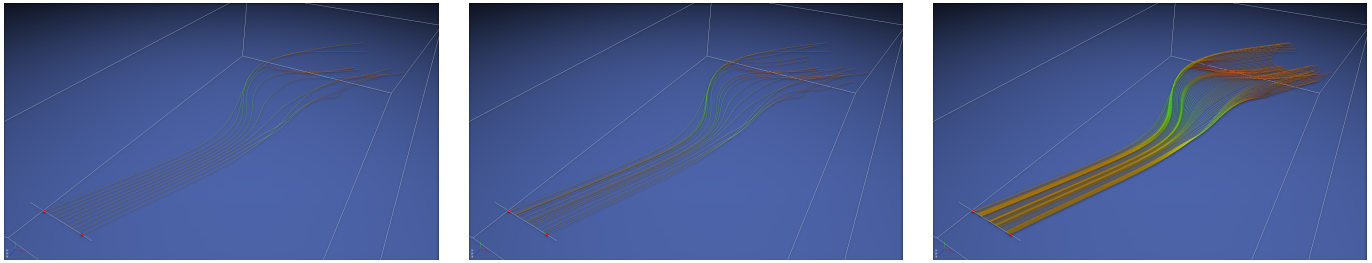
(a) Euler



(b) Cash-Karp

Figure 3: Euler and Runge-Kutta integration methods in a perfect rotation around center which are integrated with the same step size.

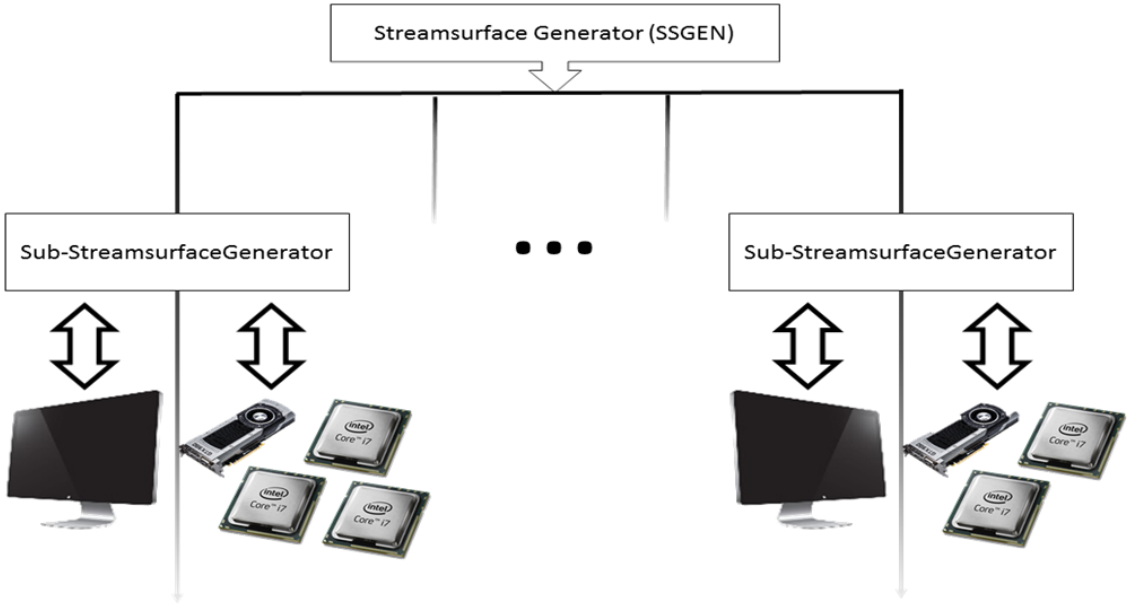
Refinement Strategy Results



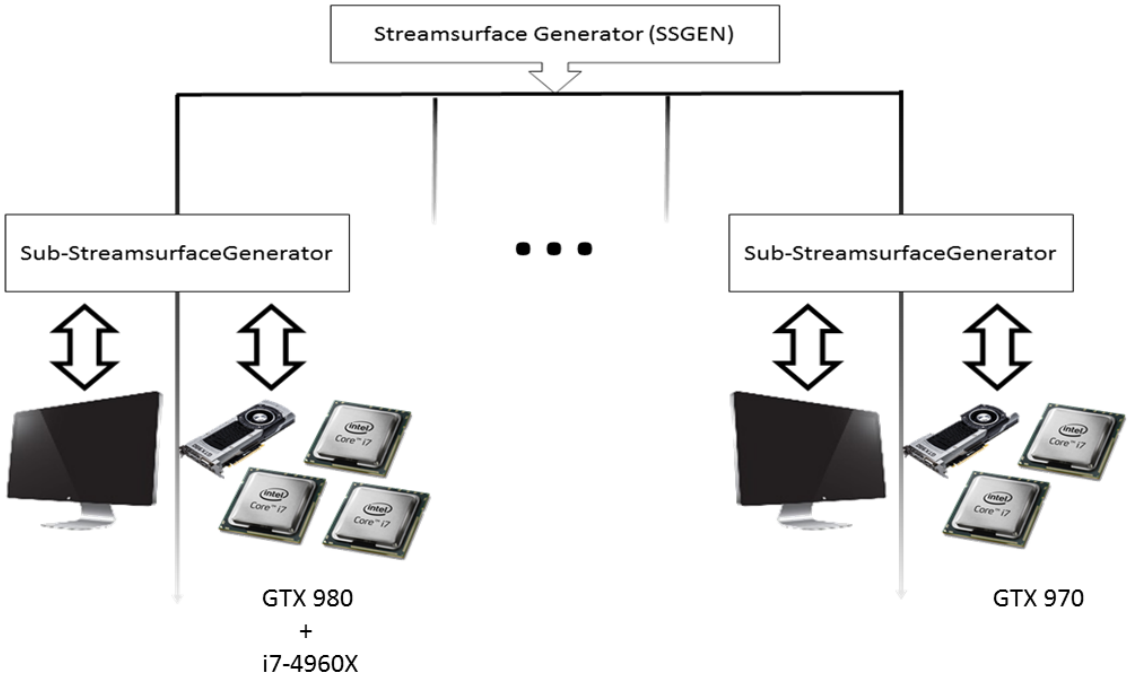
(a) No Refinements (b) The first refinement iteration (c) The second refinement iteration

Figure 4: The result streamlines making skeleton of stream surface after 2 steps refinement.

Contrib. 1: Heterogeneous Computing Arch. (1/7)



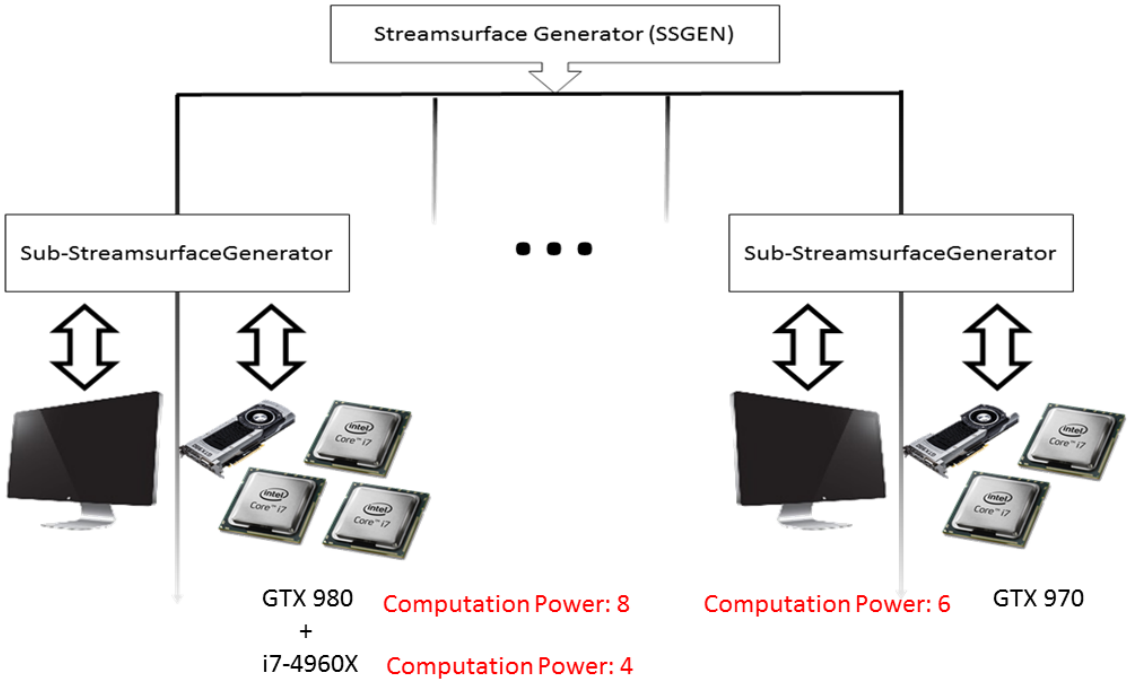
Contrib. 1: Heterogeneous Computing Arch. (2/7)



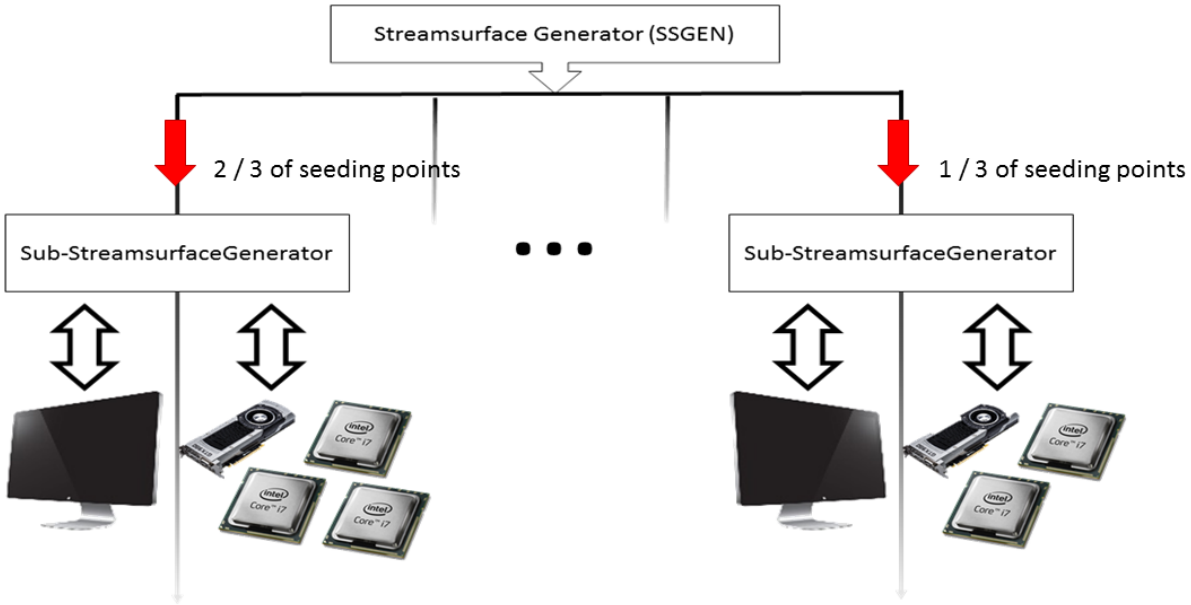
Contrib. 1: Heterogeneous Computing Arch. (3/7)

Device Name	Computation Power Weight (CPW')
NVIDIA GeForce GTX 980	8.0
NVIDIA GeForce GTX 970	6.0
Intel Core i7-4960X	4.0
TAHITI (AMD Radeon R9 280X)	6.0
PITCAIRN (AMD Radeon HD 7850)	4.5
Intel Pentium G860	1.0

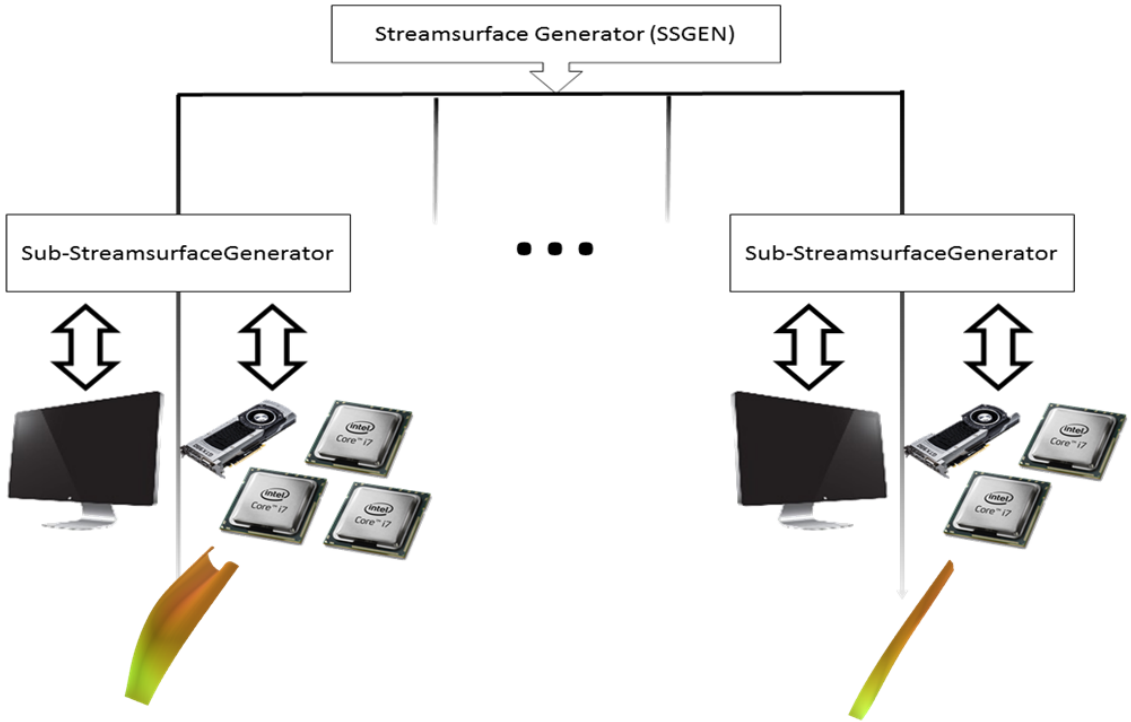
Contrib. 1: Heterogeneous Computing Arch. (4/7)



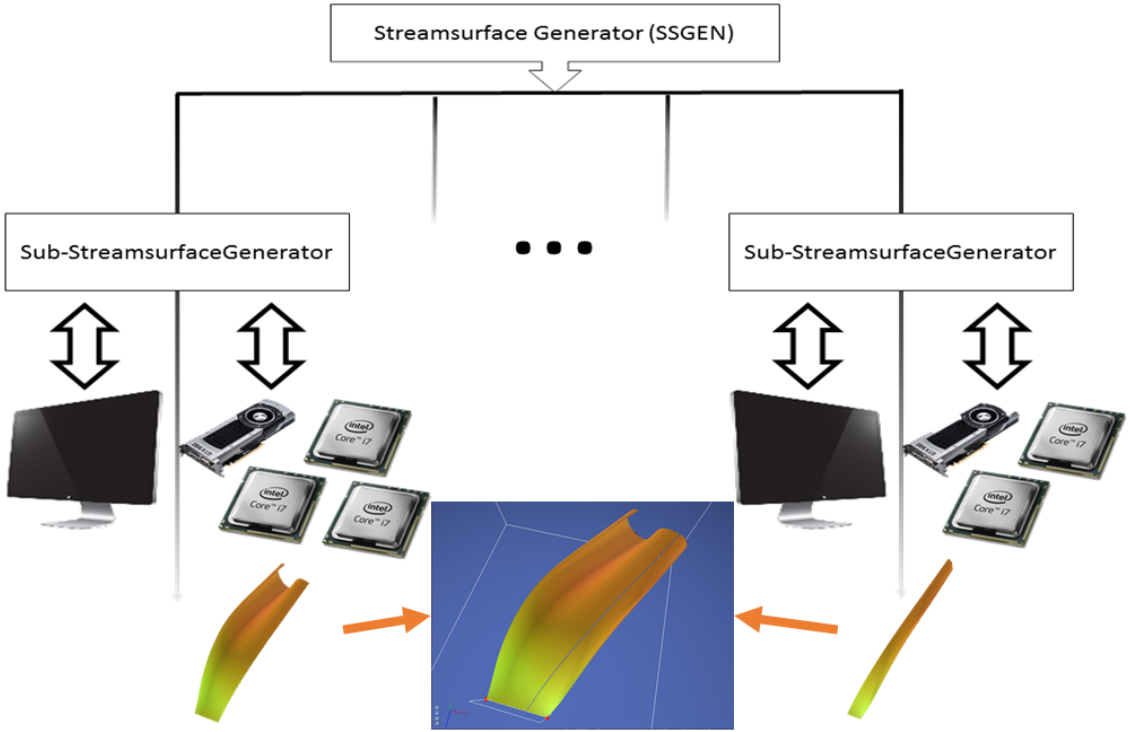
Contrib. 1: Heterogeneous Computing Arch. (5/7)



Contrib. 1: Heterogeneous Computing Arch. (6/7)

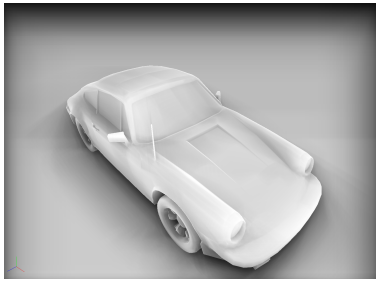


Contrib. 1: Heterogeneous Computing Arch. (7/7)

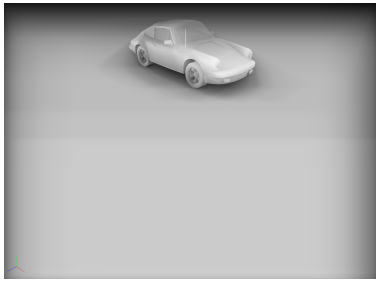


Multi GPU Rendering

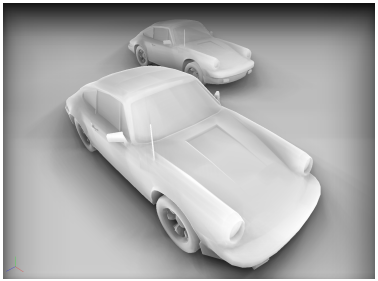
- Compositing screen cpace ambient occlusion results



(a) GPU #0

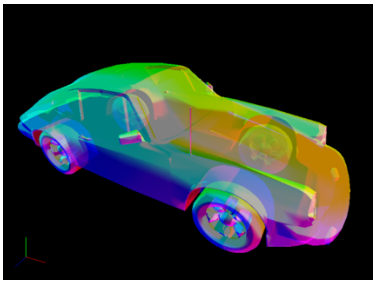


(b) GPU #1

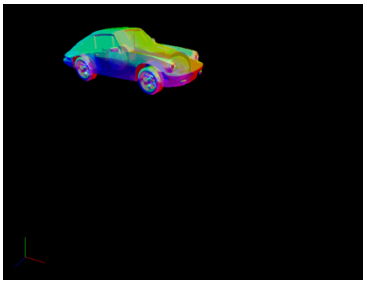


(c) Composited

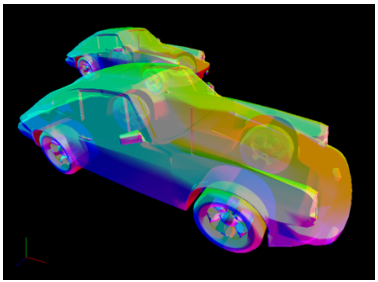
- Compositing order independence transparency results



(d) GPU #0



(e) GPU #1

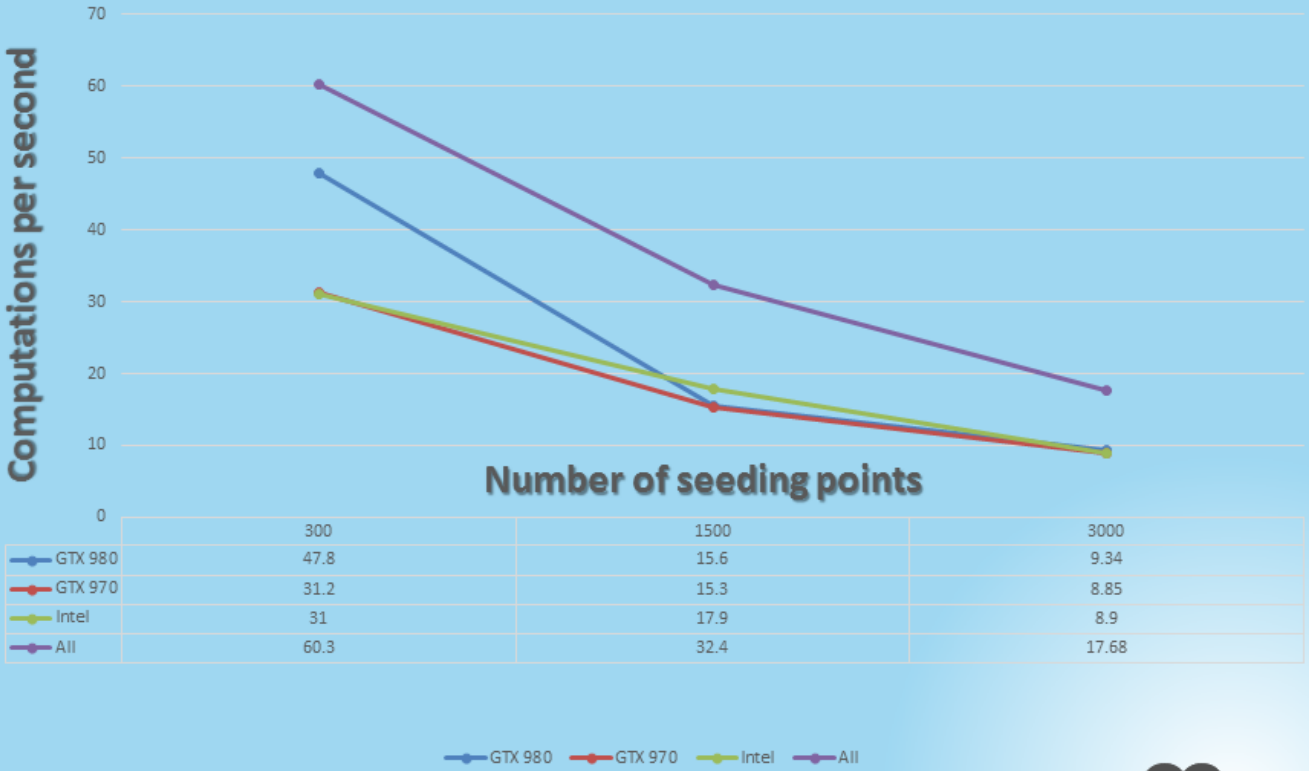


(f) Composited

Heterogeneous Computing Results

- Measured parameters:
  - **Computations per second**: It shows how many times the stream surface can be computed in every second. The unit is number of computation times per second.
  - **Number of seeding points**: Total number of seeding points for producing the final stream surface.

Wind Tunnel Dataset

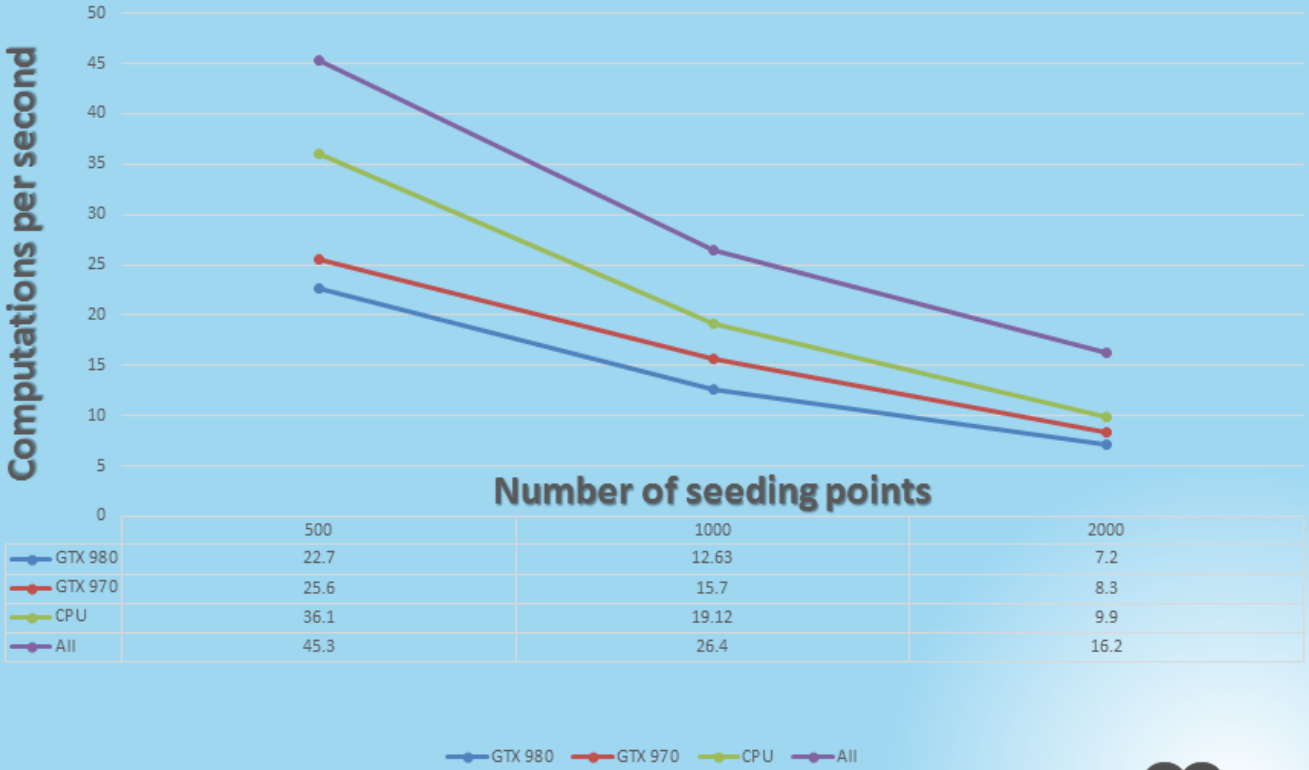


	300	1500	3000
GTX 980	47.8	15.6	9.34
GTX 970	31.2	15.3	8.85
Intel	31	17.9	8.9
All	60.3	32.4	17.68

Real-time stream surface computation – April 2016

Seyedmorteza Mostajabodaveh

Car Dataset



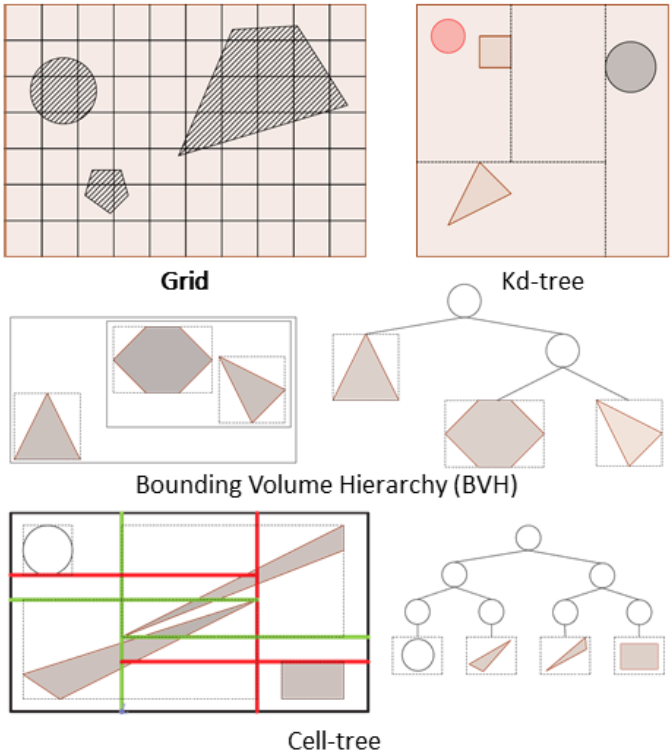
	500	1000	2000
GTX 980	22.7	12.63	7.2
GTX 970	25.6	15.7	8.3
CPU	36.1	19.12	9.9
All	45.3	26.4	16.2

Heterogeneous Computing Results Conclusions

- As expected, the streamline computation is being scaled on all available computation devices.
- Using both GPUS and CPU, the computation performance is doubled.
- Limiting factors:
  - OpenGL operations cannot be parallelized.
  - Synchronization of threads are a overhead.
  - **Dynamic task scheduling** could be a better approach.
- There is an overhead for initializing computations on multiple devices.

Acceleration Structures

- Grid
- Kd-tree
- Bounding Volume Hierarchy (BVH)
- Cell-tree (Garth and Joy, 2010)



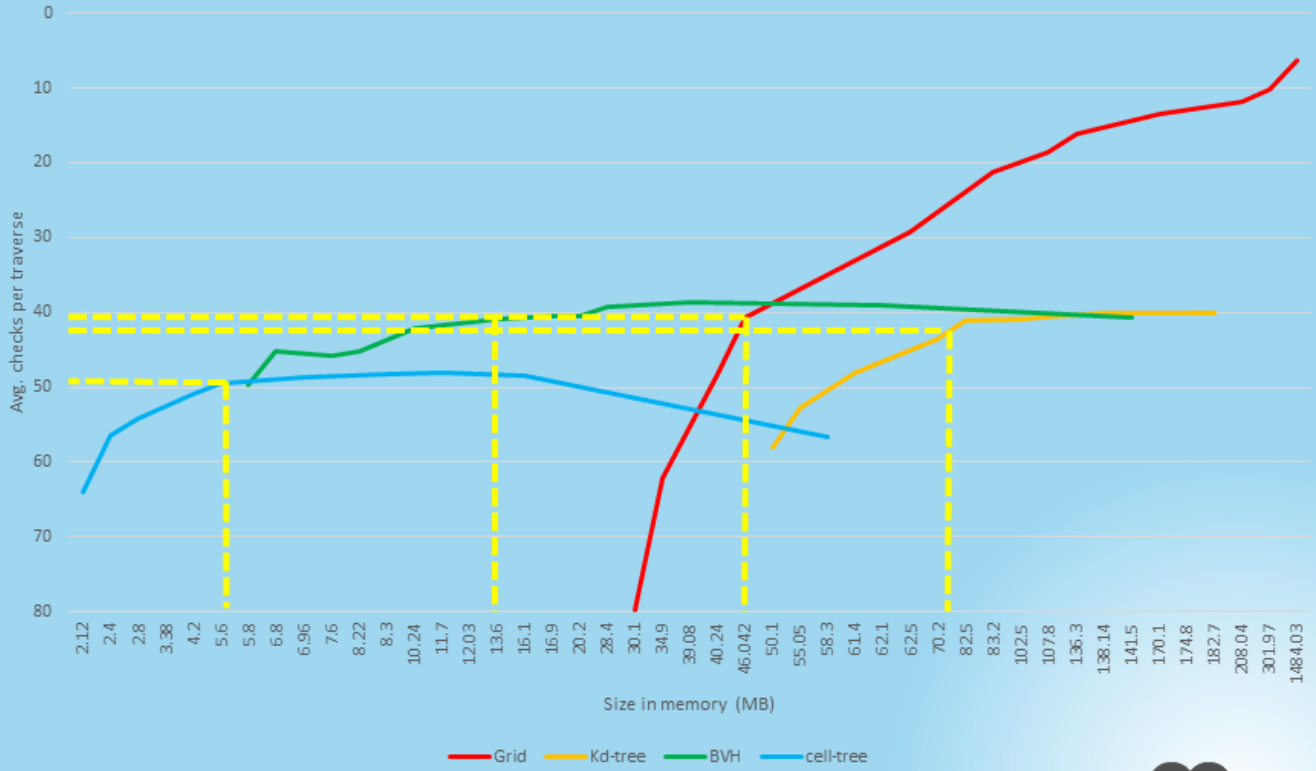
Why Acceleration Structures?

- Accelerating cell look up for faster streamline computation.
- Ray tracing uses acceleration structures to find **nearest intersection point** in contrast to streamline computation which uses it to **sample result in a specific point**.
- Differences:
  - In streamline computation instead of *Surface Area* Heuristics, textitVolume Heuristics should be used.
  - *Traversal algorithms* for space partitioning cases does not need *stack*.
  - Hierarchy traversal algorithms need a greedy method to prefer one child to other one during traversal.

Contrib. 2: Using Acceleration Structures for Streamline Computation Evaluations

- Measured parameters:
  - **Avg. checks per traverse**: Average number of bounding box or point-inside-cell tests done to find the cell surrounding the sample point.
  - **Size in memory**: Size of the constructed acceleration structured in memory.

Wind Tunnel (~2.5 M Tets, ~133 MB)



Real-time stream surface computation – April 2016

Seyedmorteza Mostajabodaveh

Wind Tunnel (~12.5 M Tets, ~800 MB)

Real-time stream surface computation – April 2016

Seyedmorteza Mostajabodaveh

Wind Tunnel (~54.1 M Tets, ~4.1 GB)

Real-time stream surface computation – April 2016

Seyedmorteza Mostajabodaveh

Acceleration Structures Evaluation Conclusion

- When memory is the bottleneck *Cell tree* is suitable (Cell tree size in memory is around 4-6 percent of original dataset).
- Highest performance is achieved using *Grid* (Grid performance is higher than BVH and cell tree when its size is higher than 25 percent of original dataset).
- BVH seems like a balance between memory and performance (BVH size in memory in its highest performance is around 9-11 percent of original dataset).

More Information

- More information can be found in original thesis slides and document (sent separately).
- Available Result Videos:
  - Stream Surface in Wind Tunnel dataset:
    `https://www.youtube.com/watch?v=DmGjogQDpyE`
  - Stream Surface in Tethrahedradecal Dataset:
    `https://www.youtube.com/watch?v=rdoRs-VpGhw`
  - Streamlines in Telescope Dataset:
    `https://www.youtube.com/watch?v=mln05oIApY0`

Outline

Ray Tracing Revisited for Rendering CSG Models Consisting of Higher Order Primitives

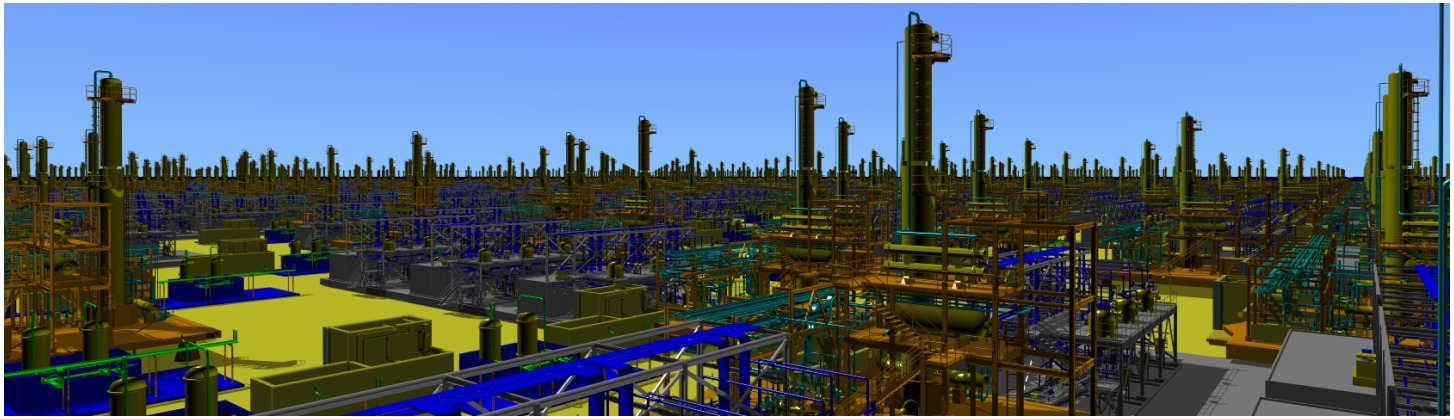


Figure 5: A complex CAD scene built out of 8,000 individual plant models. In total, the scene consists of more than 100,000,000 non-planar second-order (cylinders, cones, etc.) and forth-order (tori) primitives, ray traced at more than 10 frames per second on 16 CPU cores, including pixel-accurate shadows.

Benefits

- **On-the-fly compositing.**
  – Boolean set operations are performed on a per-ray basis immediately during rendering.
  – No Preprocessing.
- **Low storage requirement.**
  – small set of parameters for each primitive.
  – No prior triangulation of primitives. – Holding scenes in memory would be possible without triangulation.
- **Pixel-accurate higher order surfaces.**
  – Ray-primitive intersection can be tested directly. – Usually a quadratic equation which is easy to solve (tori has a quartic equation).
  – No discretization artifacts are visible.
  – Surfaces appear perfectly smooth.

Main Contributions

- a method to apply CSG operations in real-time during ray tracing while keeping the memory overhead low.
- Evaluation of the proposed method in comparison to other well-known rasterization-based real-time CSG model rendering techniques((Goldfeather et al., 1986), and (Stewart et al., 2002)).

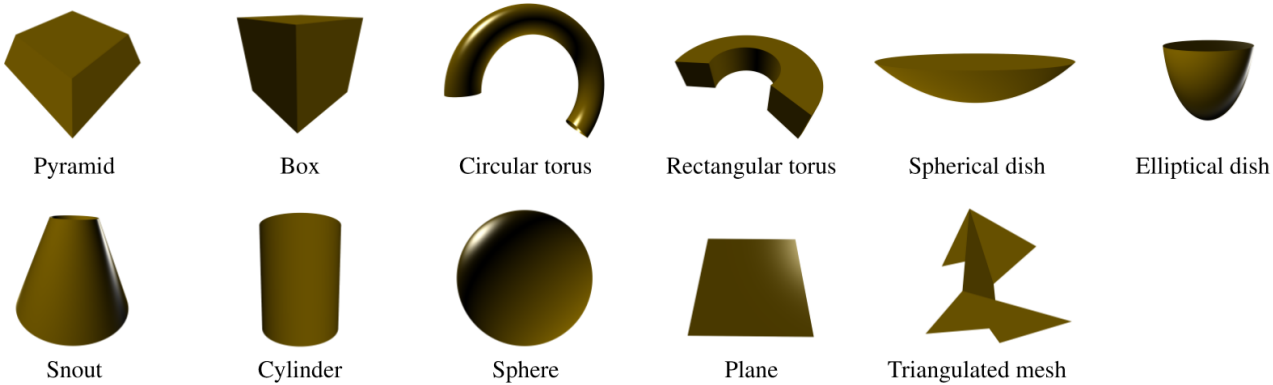
Contribution: Optimized Hitpoint Calculation

ray: current ray hitting a primitive

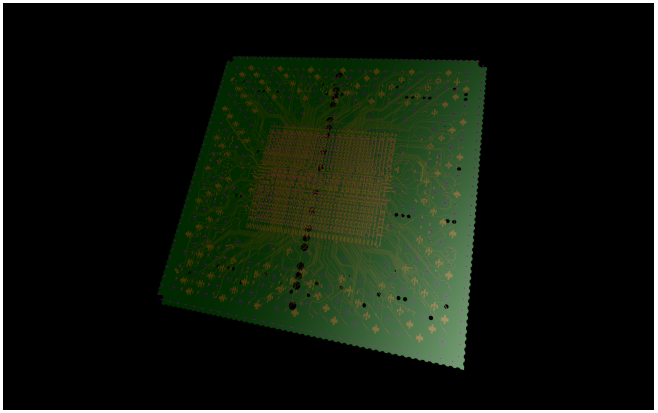
if *entering primitive* **then**
 | *delta* := +1;
else
 | *delta* := −1;
end
if *positive primitive hit* **then**
 | *ray.posDepth* += delta;
else
 | *ray.negDepth* += delta;
end
if *(ray.posDepth* > 0*)* && *(ray.negDepth* <= 0*)* **then**
 | ReportHit(); // final hit in layer found
else
 | ContinueRay(*ray*); // still inside a negative medium
end

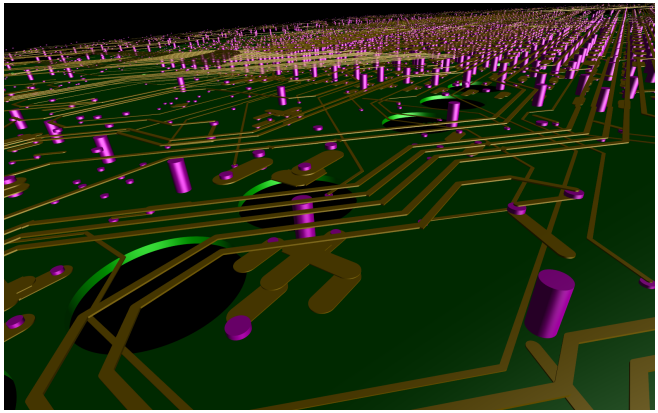
Algorithm 2: RayTraceLayer(*ray*)

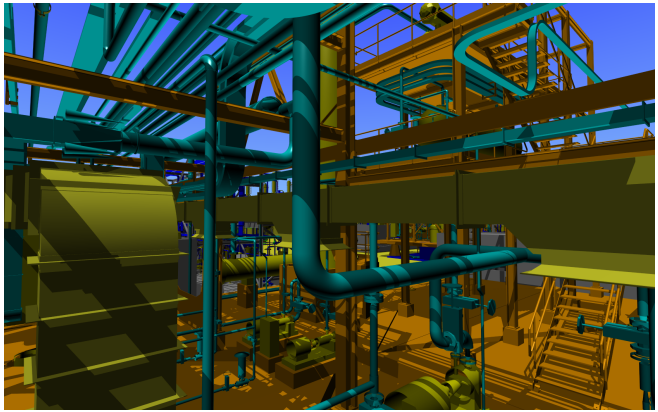
Supported Higher Order Primitives



More Results





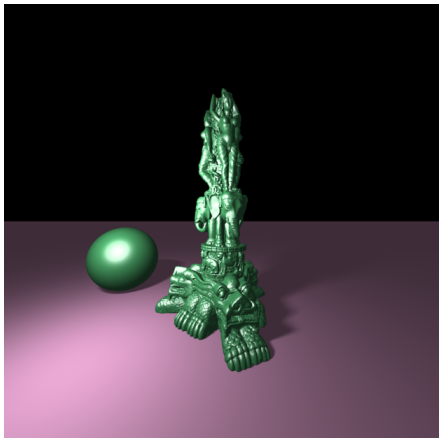


Outline

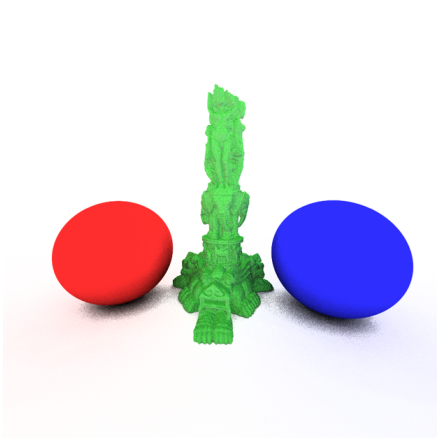
Analysis of Voxel-Based Ray Tracing

- **Goal**: Instrumenting voxel-based path-tracer with SIMD simulator to measure ray-sorting algorithms impact on SIMD units' efficiency.
- **Main Contribution**: Measure the impact of improving coherency methods (ray sorting) on instruction coherency of the rays.

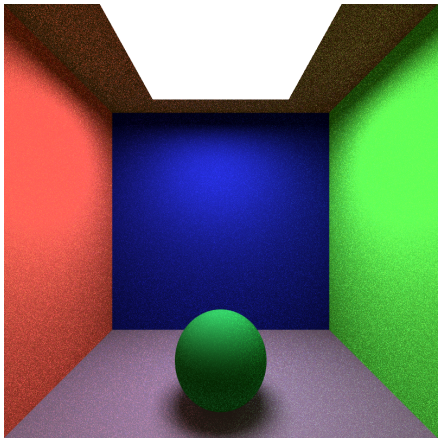
Basis for Evaluations: Implementing a Voxel-Based Path-Tracer



(e) Soft Shadow with Area Light



(f) Global Illumination with Path Tracing



(g) Cornell Box

All scenes are described with a hierarchy of 3d voxels.

BVH Traversal Main Operations Classification (1/2)

- **Push and Traverse** Ray is intersecting with both child nodes. One should be chosen and the other one should be kept in the stack.
- **Traverse** Ray is intersecting with one of the child nodes. The traversal path is clear.
- **Leaf Node Intersection** Intersection test should be done for leaf nodes.
- **Finished** Traversal is done.

BVH Traversal Main Operations Classification (2/2)

```
while Finished ≠ true do
    if CurrNode is InteriorNode then
        if intersection with both childs then
            NextNode ← leftchild
            s.Push(RightChild)                          Push and Traverse
        else
            if intersection with one child then
                NextNode ← IntersectedChild             Traverse
            else
                if not s.empty() then
                    NextNode ← s.Pop()
                else
                    Finished ← true                     Finished
                end
            end
        end
    else
        Intersect with voxels related to the leafnode to find the
         intersection.                                  Leaf Node Intersection
        if found or s.empty() then
            Finished ← true                             Finished
        else
            NextNode ← s.Pop()
        end
    end
end
```

Our SIMD Simulator

Reset all ray's *InstCounter* elements to zero.
CurrStage $\leftarrow 0$
while *Finished* $\neq$ *true* **do**
  if *All rays has done their first CurrStage operations* **then**
    CurrStage++
    Continue
  end
  CurrOp $\leftarrow$ *FindNextOperationToExecute*(CurrStage)
  ExecuteSIMDUnitsWithNextOperation(*CurrOp*)
  if *Any of SIMD Units are done* **then**
    deactivate the SIMD operation unit.
  end
end

Example of SIMD Simulation (Instruction Coherency)

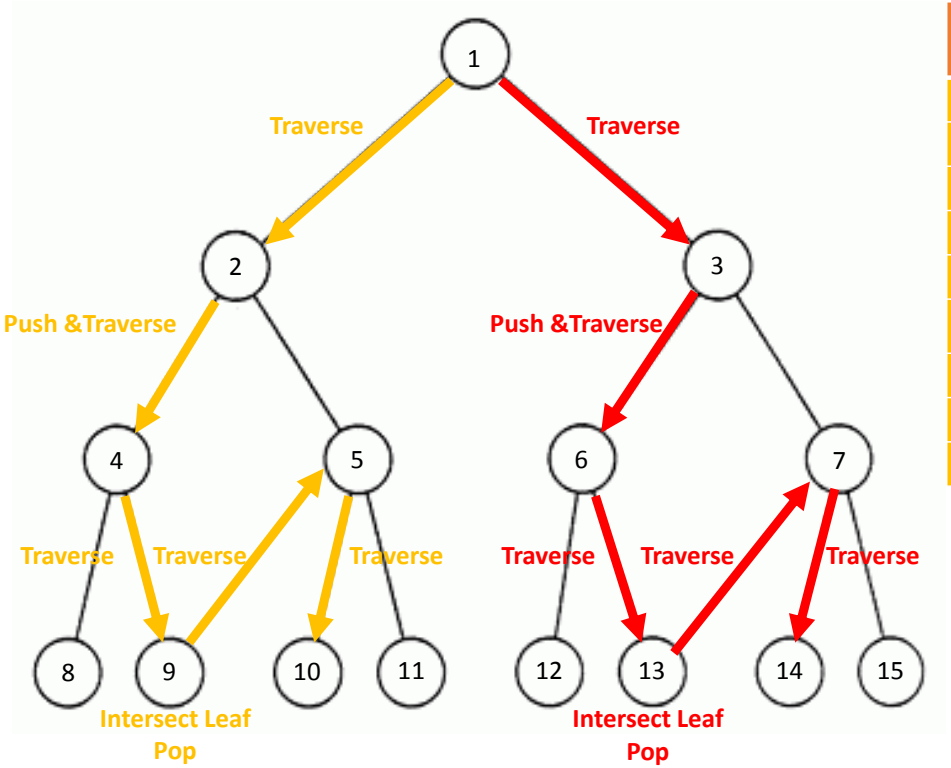
Ray 0: T, P&T, T, IL, T, Done
Ray 1: T, T, T, Done
Ray 2: T, T, Done
Ray 3: T, P&T, T, IL, T, Done

T: Traverse
P&T: Push & Traverse
IL: Intersect Leaf

		Oper 0	Oper 1	Oper 2	Oper 3	Oper 4	Oper 5
Ray 0	OpUnit 0	T	P&T	T	IL	T	Done
Ray 1	OpUnit 1	T		T		T	Done
Ray 2	OpUnit 2	T		T			Done
Ray 3	OpUnit 3	T	P& T	T	IL	T	Done
	Coherency	100%	50%	100%	50%	75%	100%

Figure 6: An example of traversing the BVH-tree for 4 different rays on SIMD of width four.

Instruction Coherency Vs. Data Coherency



Instruction Coherency		
Ray1	Ray2	Coherency
T	T	100%
T	T	100%
P&T	P&T	100%
T	T	100%
IL & Pop	IL & Pop	100%
T	T	100%
T	T	100%
D	D	100%

T: Traverse
P&T: Push & Traverse
IL & Pop: Intersect Leaf & Pop
D: Done

Data Coherency		
Ray1	Ray2	Coherency
1	1	100%
2		50%
	3	50%
4		50%
	6	50%
9		50%
	13	50%
5		50%
	7	50%
10		50%
	14	50%

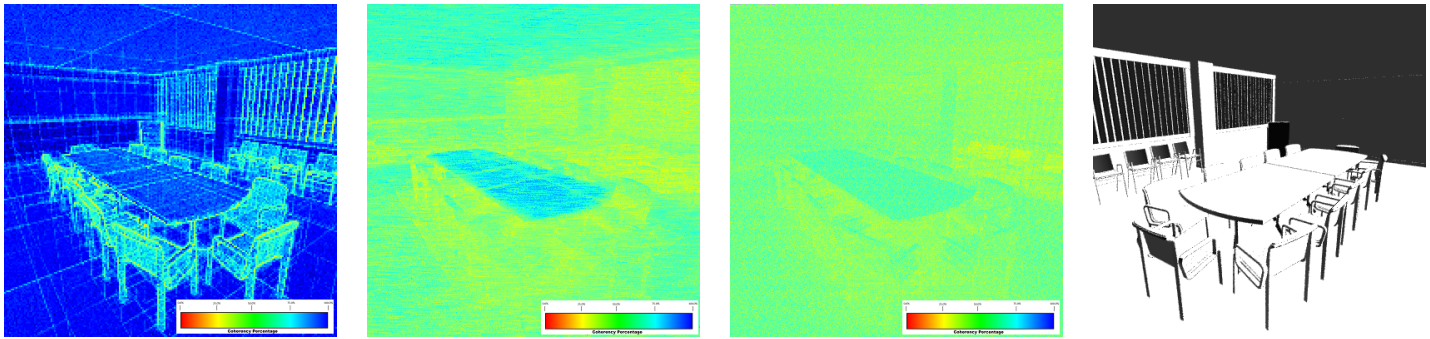
Figure 7: While both rays are perfectly instruction execution coherent, they are incoherent in data access.

Contribution: Coherency Measurements Results for Path-Tracing "Conference Room" (1/2)

Case	Primary Rays	Bounce 1	Bounce 2
No Ray Sorting	79.14%	49.53%	45.63%
Ray Sorting by Direction	79.14%	49.70%	48.3%
Ray Sorting by Position	79.14%	46.68%	45.81%
Ray Sorting by Both	79.14%	49.73%	48.6%

Table 1: Total Ray Coherency For Different Bounces of Rays (SIMD width = 16) - Conference Room

Contribution: Coherency Measurements Results for Path-Tracing "Conference Room" (2/2)



(a) Primarys (b) Bounce 1 (c) Bounce 2 (d) Result

Figure 8: Coherency textures for the case which rays are sorted based on their direction and orientation on every bounce (SIMD width = 16). Colormap used for visualizing coherency is shown in the bottom-right corner of the figures. The range is from 0-100 percent (red-blue).

Similar Papers

- (Barringer and Akenine-Möller, 2014), and (Chajdas and Westermann, 2014)
- They also show that while ray sorting algorithms improve ray tracing performance, they are not very effective, i.e. hopefully new approaches can be devised to make voxel-based ray tracing even faster.

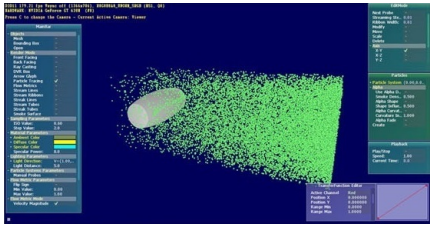
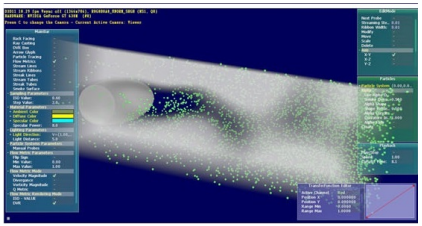
Outline

Practical Courses

- Interactive Visual Data Analysis, Prof. Westermann Group.
- GPU Programming in Computer Vision, Prof. Cremers Group.
- Guided Research: Voxel-Based Path tracing, Prof. Westermann Group.
- Interdisciplinary Project: 3D-Visualization of electromagnetic field strength distribution, Prof. Guenthner (Fördertechnik Materialfluss Logistik group in TU-Munich).

Interactive Visual Data Analysis

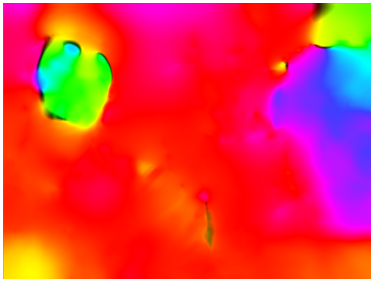
- Fundamental knowledge in the field of scientific visualization:
  - Volume Visualization
  - Flow Visualization
  - GPGPU Programming
  - Done with C++, D3D11, and HLSL.
- See results as a video on:
  `https://www.youtube.com/watch?v=7pIw5PjkjmY`


GPU Programming in Computer Vision

- Implementing *Optical Flow* on GPU



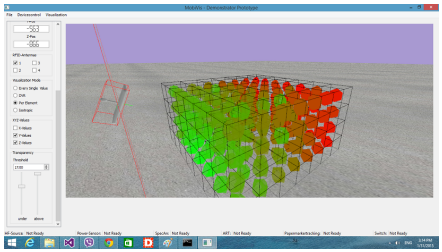


- Implementing *Super Resolution* on GPU

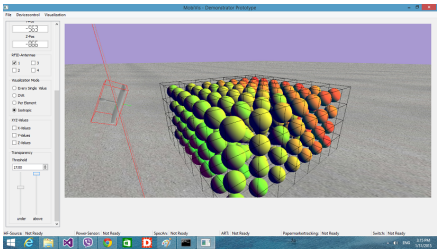

Interdisciplinary Project: Implementing a Framework for 3D Visualization of Electromagnetic Field Strength Distribution

- Build a model of examined installation in 3d virtual world.
  - fast and intractable (Using tracking system)
- Visualizing 3d bars in virtual world to specify amount of data collected in each area.
- Visualizing collected data using different techniques.
  - Goal: Detecting weak/strong magnetic fields in different parts of environment.
  - Visualizing data using torus and sphere glyphs.
  - Visualizing data using direct volume rendering.

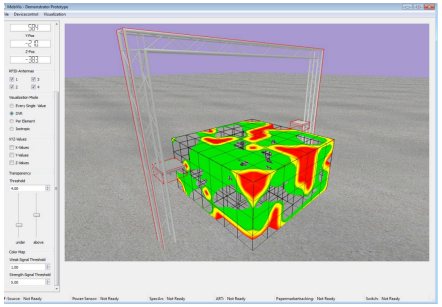
Interdisciplinary Project: Implementing a Framework for 3D Visualization of Electromagnetic Field Strength Distribution



(i) Torus Glyph



(j) Sphere Glyph



(k) Direct Volume Rendering

Figure 9: Different visualization techniques used in this project (screenshots are not captured in for the same data).

Related Courses

- Computer Graphics related: Computer Graphics, Image Synthesis, Scientific Visualization, Augmented Reality, Computer Animation and Simulation.
- Parallel Programming related: Parallel Programming, Advanced Computer Architecture.

Outline

Current Status

- Working at Fraunhofer IGD, Darmstadt as a Researcher.
- Involved in two projects:
  - **VELaSSCo EC Funded Project** (`http://www.velassco.eu/`).
    – Our simulation models are stored in a Hadoop database (HBase) which is running on a HPC.
    – Idea: Doing post processing (mesh simplification, and streamline computation) on the HPC and send the result as triangle mesh for the client.
    – My task is to implement Fraunhofer IGD's visualization client, some of data access queries on HPC side, and streamline computation post processing technique on the HPC side using map-reduce.
  - an industrial project in collaboration with CST AG.
    – Worked on dynamic tessellating arc primitives in real-time on CPU and upload the triangles to the GPU.
  - **Both projects will are already finished and it's a great time to leave Fraunhofer IGD for me**

References I

Barringer, R. and Akenine-Möller, T. (2014).
Dynamic ray stream traversal.
ACM Trans. Graph., 33(4):151:1–151:9.

Buerger, K., Ferstl, F., Theisel, H., and Westermann, R. (2009).
Interactive streak surface visualization on the gpu.
IEEE Transactions on Visualization and Computer Graphics, 15(6):1259–1266.

Camp, D., Childs, H., Garth, C., Pugmire, D., and Joy, K. I. (2012).
Parallel stream surface computation for large data sets.
In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 39–47.

References II

Chajdas, M. G. and Westermann, R. (2014).
Quantitative Analysis of Voxel Raytracing Acceleration Structures.
In Keyser, J., Kim, Y. J., and Wonka, P., editors, *Pacific Graphics Short Papers*. The Eurographics Association.

Garth, C. and Joy, K. (2010).
Fast, memory-efficient cell location in unstructured grids for visualization.
Visualization and Computer Graphics, IEEE Transactions on, 16(6):1541–1550.

References III

Goldfeather, J., Hultquist, J. P., and Fuchs, H. (1986).
Fast Constructive-Solid Geometry Display in the Pixel-Powers Graphics System.
In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, pages 107–116.

Schirski, M. (2008).
Interactive particle tracing for the exploration of flow fields in virtual environments.

Stewart, N., Leach, G., and John, S. (2002).
Linear-time csg rendering of intersected convex objects.
In *In 10th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision - WSCG 2002 (2002*, pages 437–444.