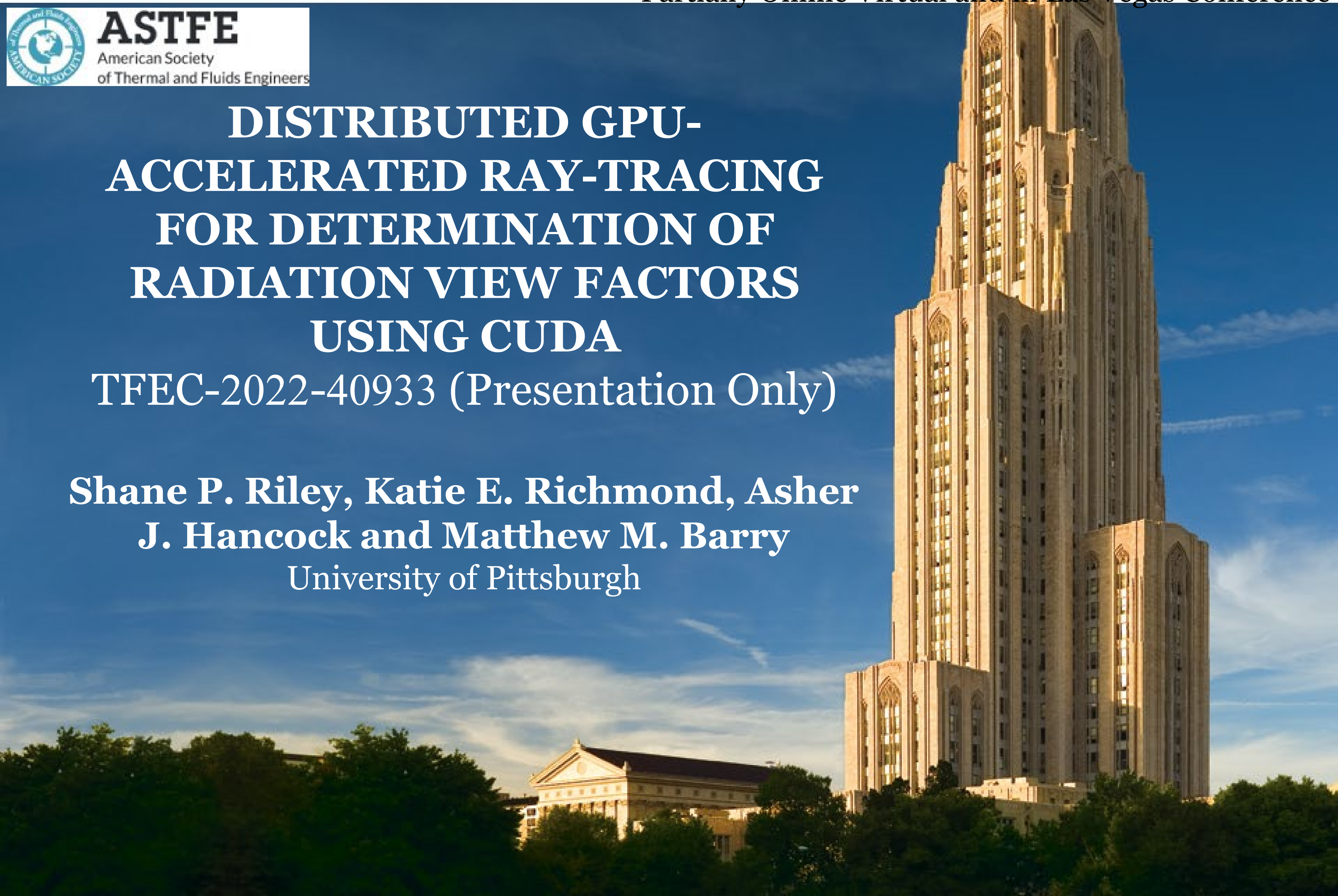
7th Thermal and Fluids Engineering Conference (TFEC)
May, 16-18, 2022
Partially Online Virtual and in Las Vegas Conference

University of Pittsburgh

ASTFE
American Society of Thermal and Fluids Engineers



DISTRIBUTED GPU-ACCELERATED RAY-TRACING FOR DETERMINATION OF RADIATION VIEW FACTORS USING CUDA

TFEC-2022-40933 (Presentation Only)

Shane P. Riley, Katie E. Richmond, Asher J. Hancock and Matthew M. Barry
University of Pittsburgh

Background: View factor

Radiation view factor (F_{ij}):

$$F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j}\frac{\cos(\theta_i)\cos(\theta_j)}{\pi\|\overrightarrow{R}_{ij}\|^2}\,dA_i\,dA_j$$

A geometric property that quantifies the proportion of diffuse radiation emitted from one surface, A_i, and received by another surface, A_j

Radiation Heat Transfer Rate (Q_i):

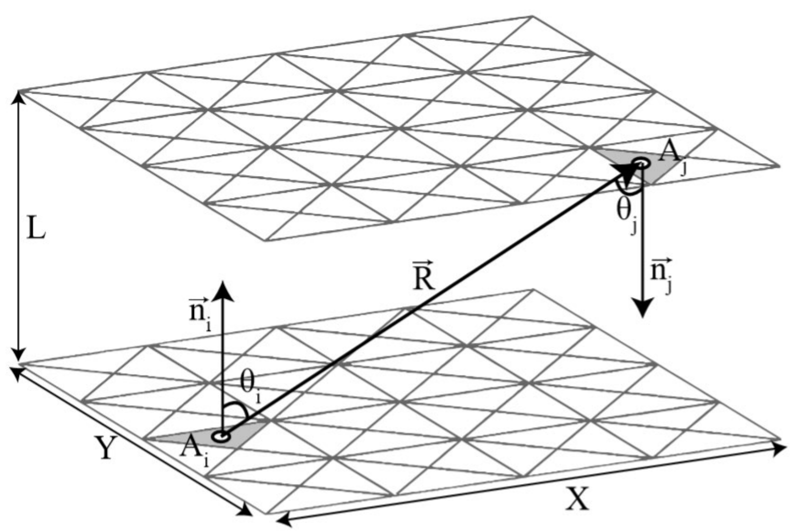
$$Q_i = \varepsilon\sigma A_i F_{ij}\left(T_i^{\,4} - T_j^{\,4}\right)$$

ε : material emissivity
σ: Stefan-Boltzmann constant
T_i: temperature of emitter
T_j: temperature of receiver

Why do we care?

Accurate heat transfer models

Parasitic radiative transfer that lowers temperature gradient

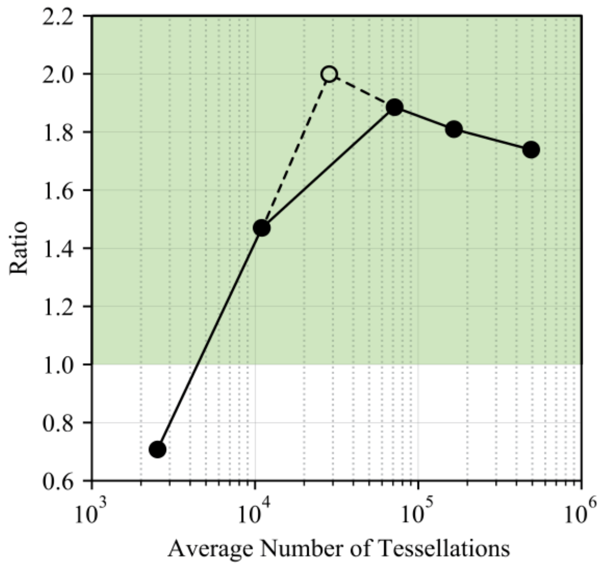


Motivation: Previous Work

Hancock et. al. implemented a numeric view-factor solver using Aparapi, a library to convert JAVA bytecode to OpenCL GPU code

While the Aparapi code outperforms CPU and scales to multiple GPUs, the speedup factor decays when large numbers of tessellations are applied

The successes and drawbacks of the Aparapi code necessitates a new implementation



		Time [s]		
Avg. Tess.	F_{ij}	CPU	1 GPU	2 GPU
2,527	5.759450587e-4	5.505	0.955	1.350
11,015	5.751577438e-4	16.13	2.764	1.881
71,586	5.754201918e-4	113.5	62.69	33.25
165,291	5.754852079e-4	472.8	274.7	151.8
491,186	5.7545765976e-4	4,160	2,387	1,372

Methodology: GPU-accelerated programming and geometry definition

Graphics Processing Unit (GPU) can achieve massive runtime gains

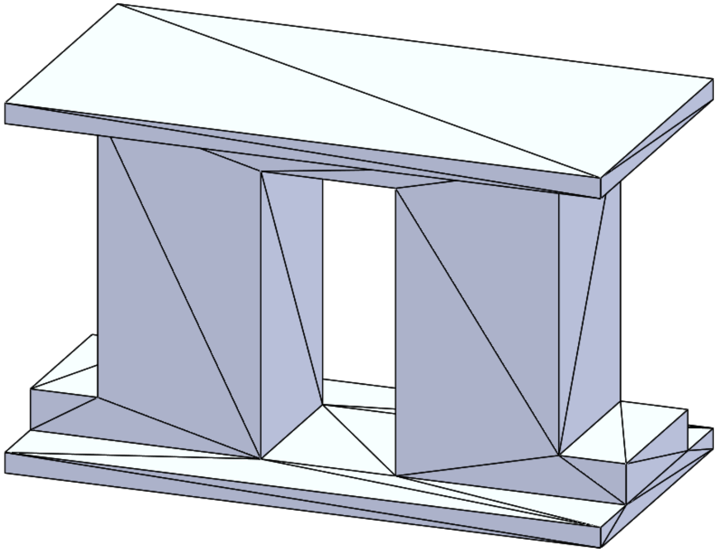
- Thousands of cores that operate in parallel
- Computational structure is ideal for repetitive, arithmetic tasks

Geometry Input: STL Files

- Ubiquitous in CAD software
- Robust geometry definition

```
solid ThermoelectricGenerator.STL
   facet normal 0.000000e+00 -1.000000e+00 0.000000e+00
      outer loop
         vertex -4.128709e-01 1.750000e+00 5.000000e-01
         vertex -4.128709e-01 1.750000e+00 -5.000000e-01
         vertex 4.128709e-01 1.750000e+00 5.000000e-01
      endloop
   endfacet
```

Example ASCII STL format



Example TEG unit-cell defined in STL format

Methodology: View factor computation

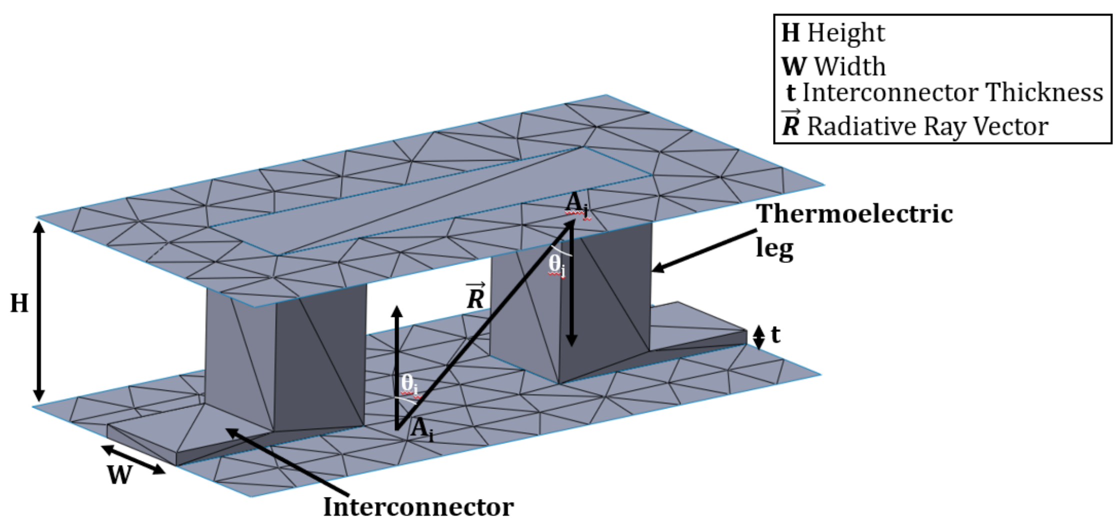
View factor:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos(\theta_i)\cos(\theta_j)}{\pi \| \overrightarrow{R}_{ij} \|^2} \, dA_i \, dA_j$$

View factor summation:

$$F_{ij} = \frac{1}{A_i} \sum_{i=1}^{N_i} \sum_{j=1}^{N_j} \frac{\cos(\theta_i)\cos(\theta_j)}{\pi \| \overrightarrow{R}_{ij} \|^2} \, dA_i \, dA_j$$

Every differential area of the emitter surface creates a corresponding ray for each receiver differential area, where the ray vectors are determined by the centroidal locations of each triangle, as determined from the STL file.

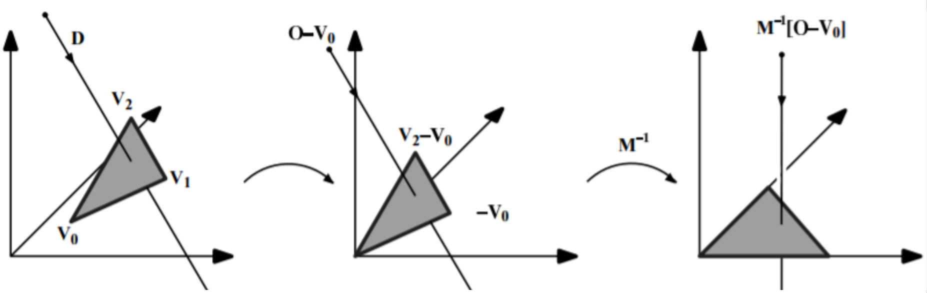


Methodology: Shadow effect and Möller–Trumbore intersection algorithm

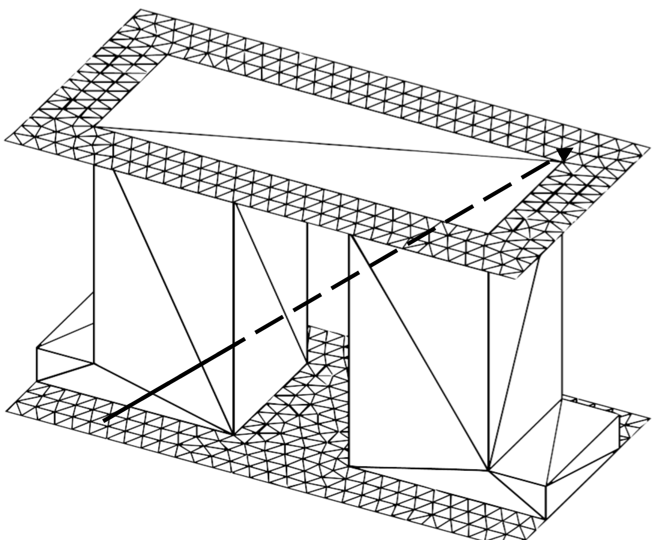
Shadow effect: a phenomenon that represents any potential ray intersection with a non-participating surface

Reduces view factor magnitude

Möller–Trumbore ray-triangle intersection algorithm:

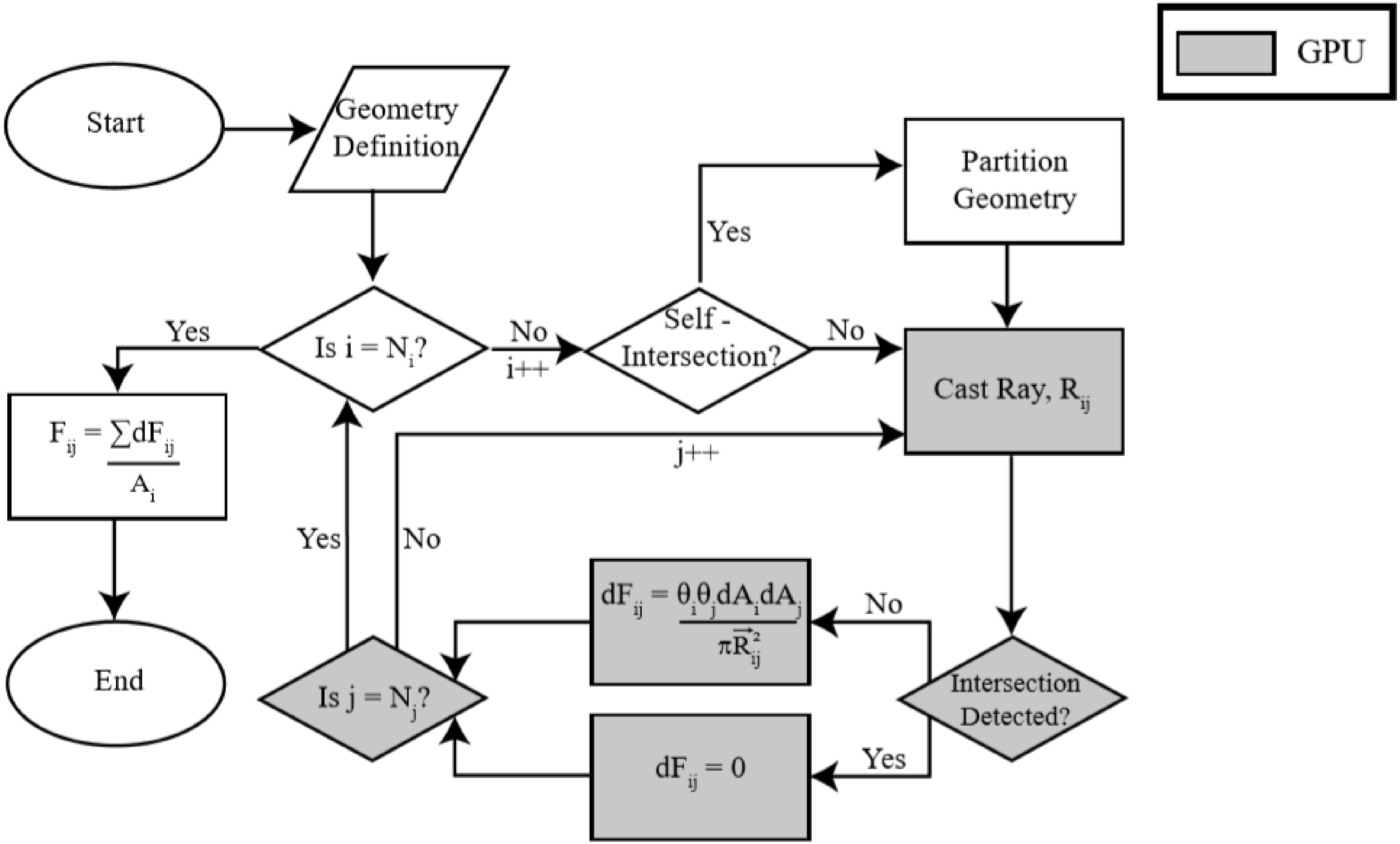


“Fast, minimum storage ray-triangle intersection” (1997) in Journal of graphics tools



Example of shadow effect

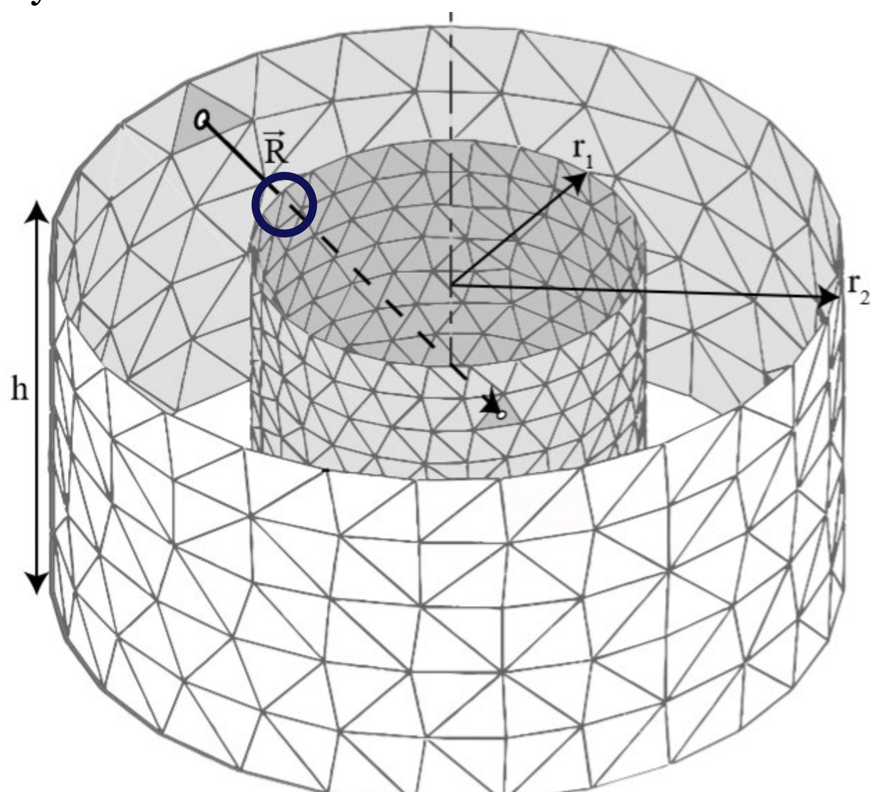
Methodology: Flow chart



Methodology: Self-intersection

Self-intersection : refers to any obstructive surface, intrinsic of either emitting or receiving surface, that need checked by the MT algorithm for possible ray-intersection to properly resolve the view factor

Normally exhibited in curved surfaces

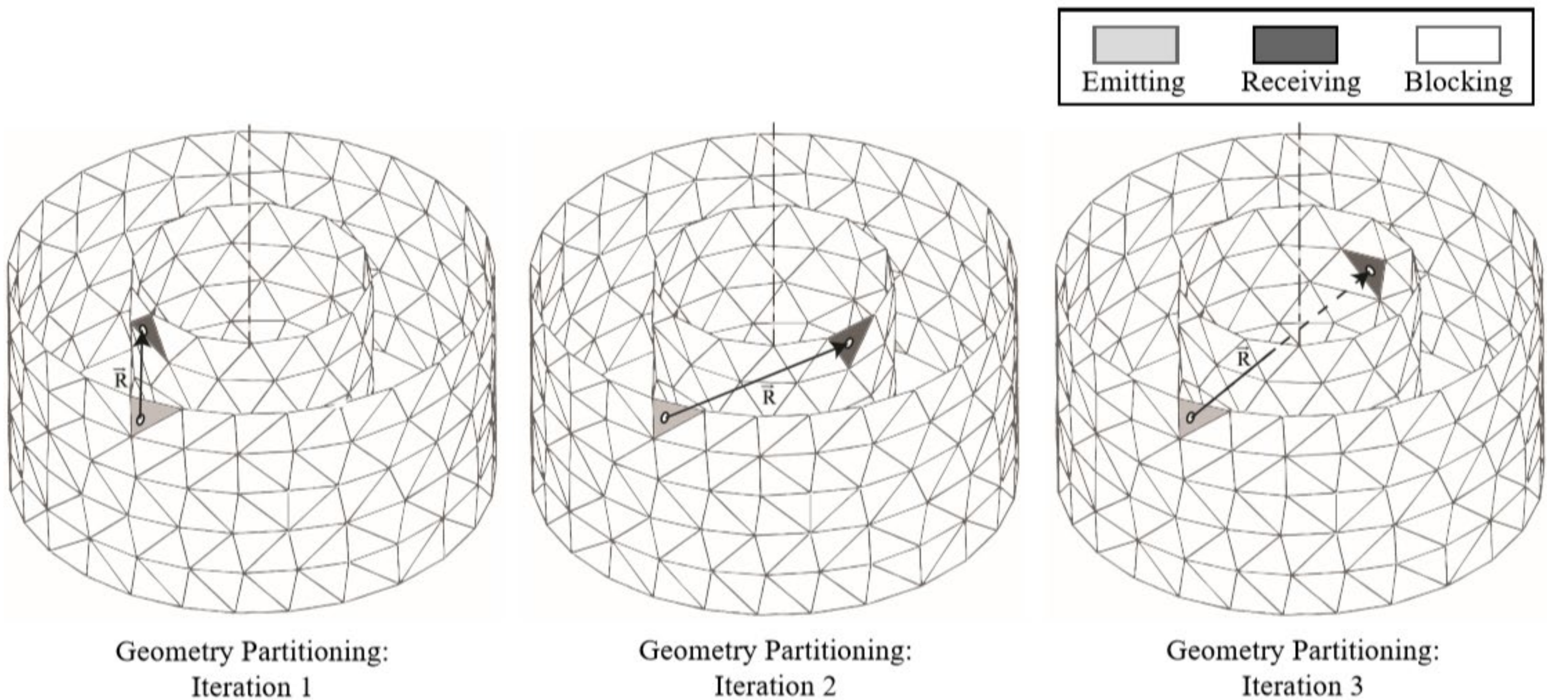


Concentric cylinders that exhibit self-intersection

Methodology: Self-intersection algorithm

Self-intersection algorithm: dynamics updates the emitting/receiving surfaces to consider only one tessellation at a time

All other surfaces are considered non-participatory (blocking)



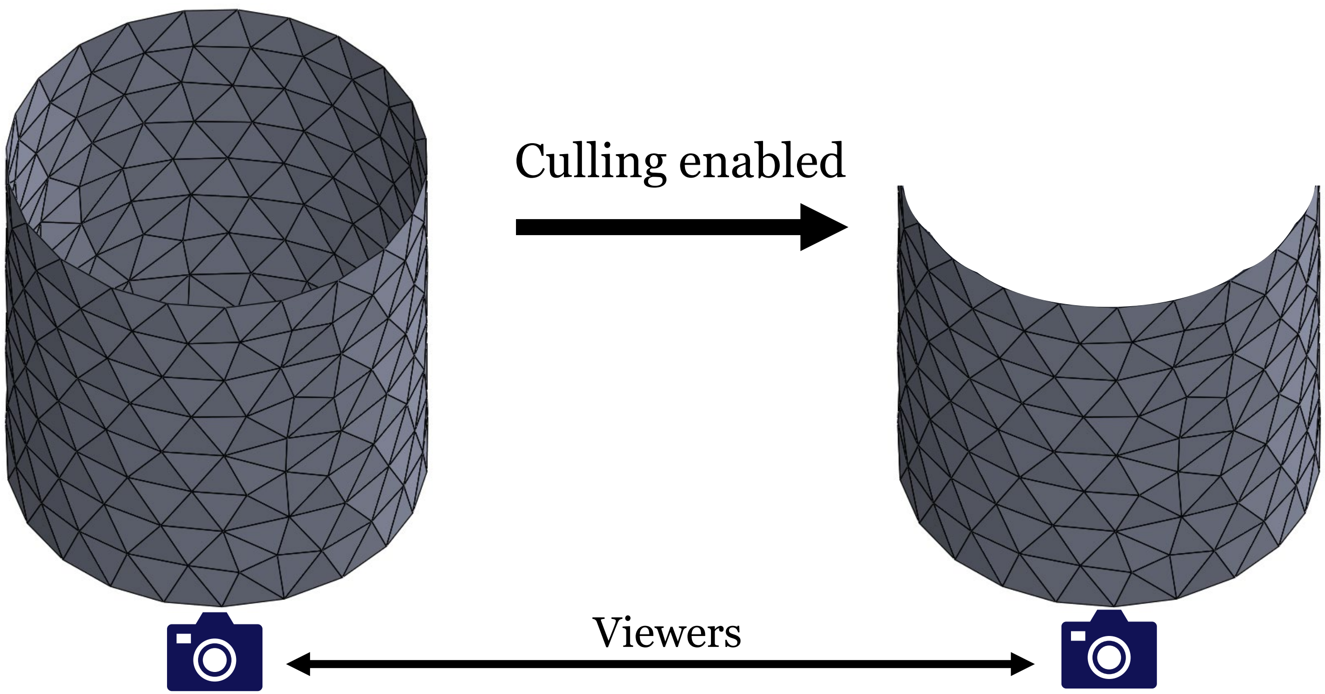
Dynamic geometry partitioning during self-intersection algorithm

Methodology: Back-face culling

Back-face culling: a computer graphics technique that refers to the removal of primitive geometries that face away from the camera

In this context, tessellations that “face away” from the emitting tessellation appear clockwise oriented

Increases computational savings and it prevents intersection miscalculations.

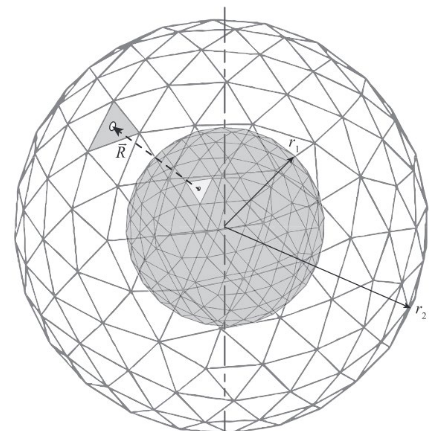


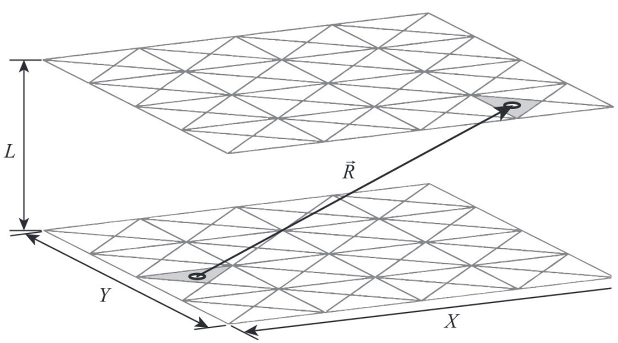
Validation:

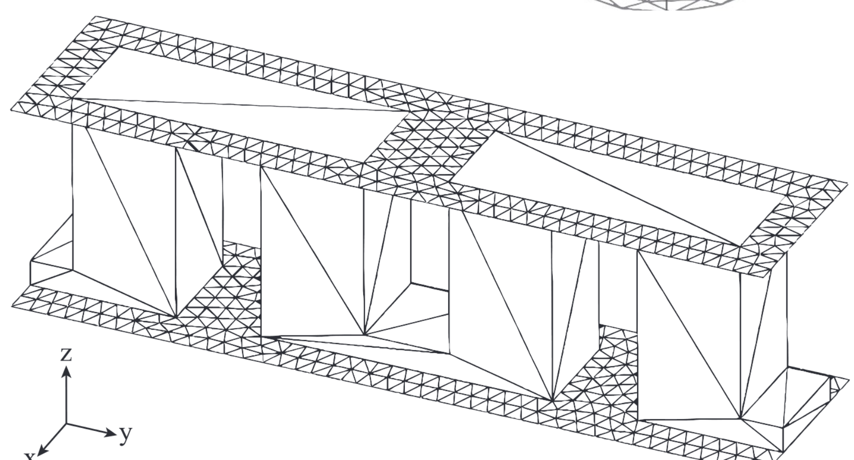
Small Validation Set:

- Parallel Plates (5 tessellation counts, ASCII, RTX 2070)
- Spheres (4 tessellation counts, ASCII, RTX 2080ti)
- TEG Unicouple (5 tessellation counts, BINARY, RTX 2070)

Each case is run on 1 and 2 cards separately
Results compared with true values







Outline:
Background
Motivation
Methodology
Validation
Results
Discussion
Conclusion

Results:

View factor approach true analytic values

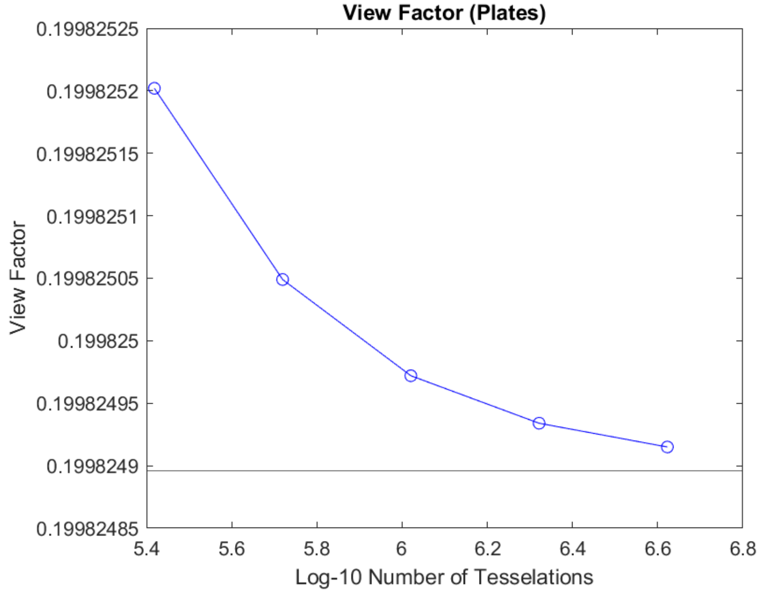
Does not reach double-precision limit, even with very high runtime

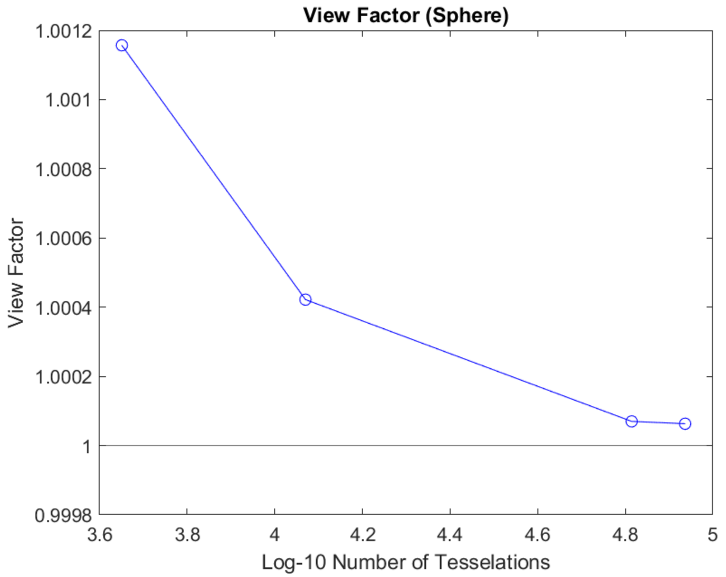
Case	CUDA VF	Analytic VF	Absolute Error	Relative Error
Parallel Plates	0.199824915	0.1998248957	1.93e-8	9.66e-8
TEG	0.000575456	N/A	N/A	N/A
Spheres	1.000063232	1.0000000000	6.32e-5	6.32e-5

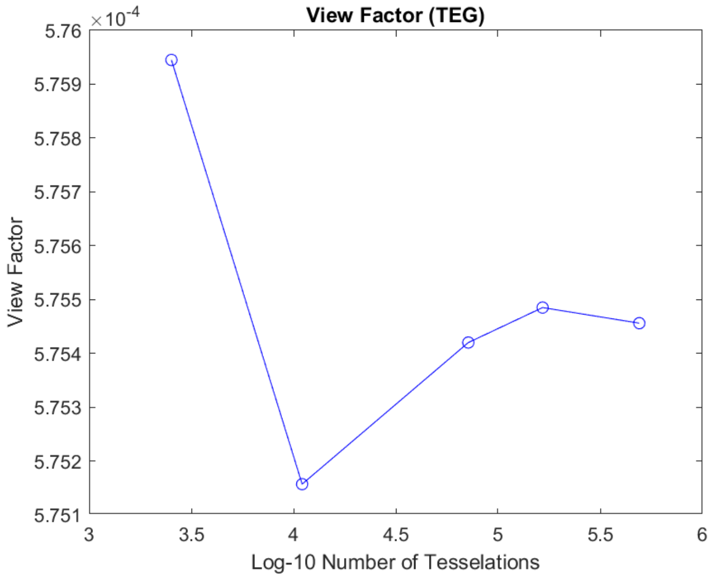
Discussion:

Result not dependent on number of GPU's

Black lines refer to analytical solution (not present for TEG)

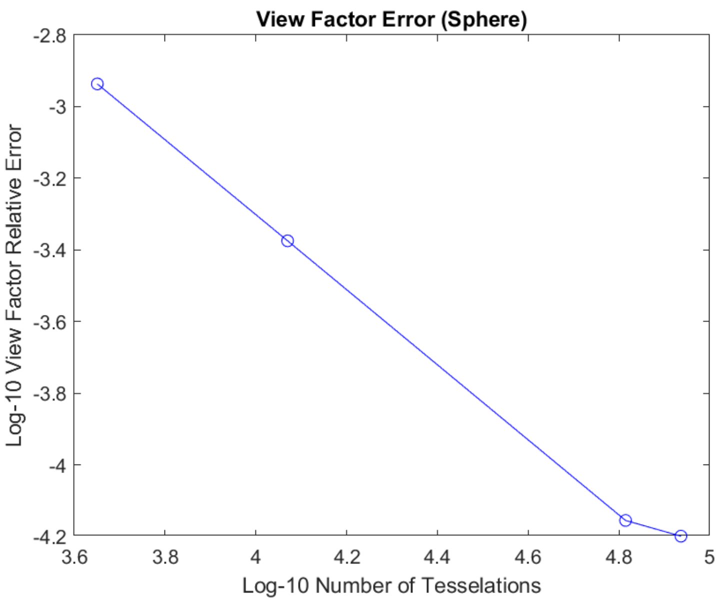


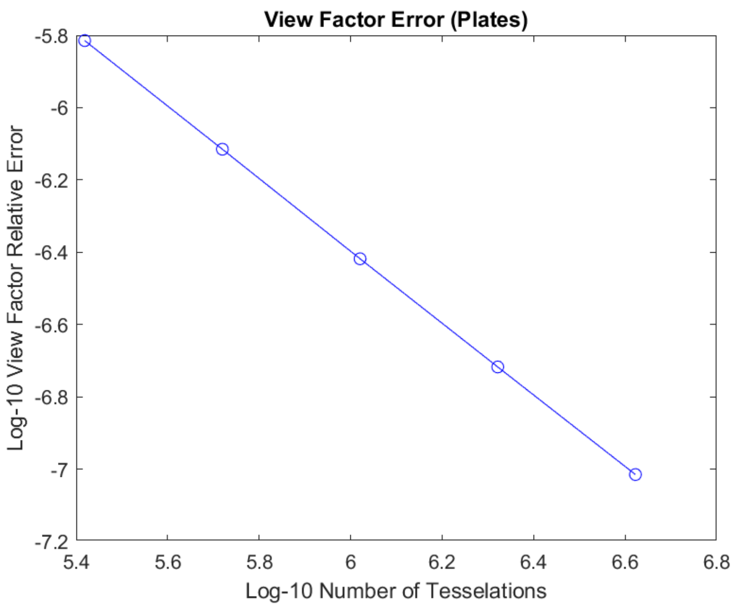




Discussion:

Result not dependent on number of GPU's

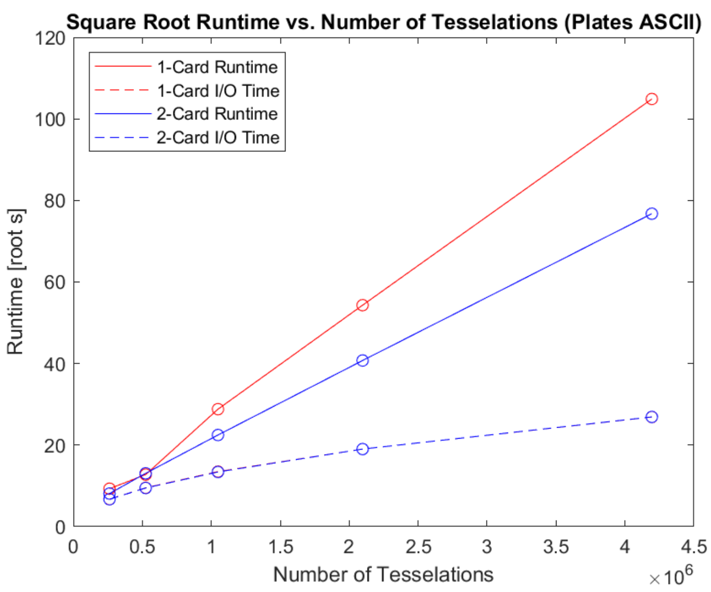


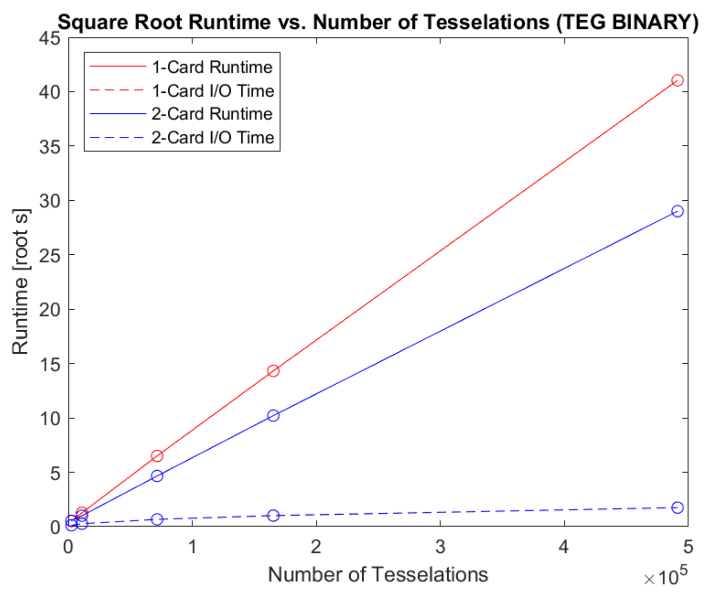


Discussion:

Due to two-loop nature of non-self-intersecting code, $O(n^2)$ asymptotic runtime is expected

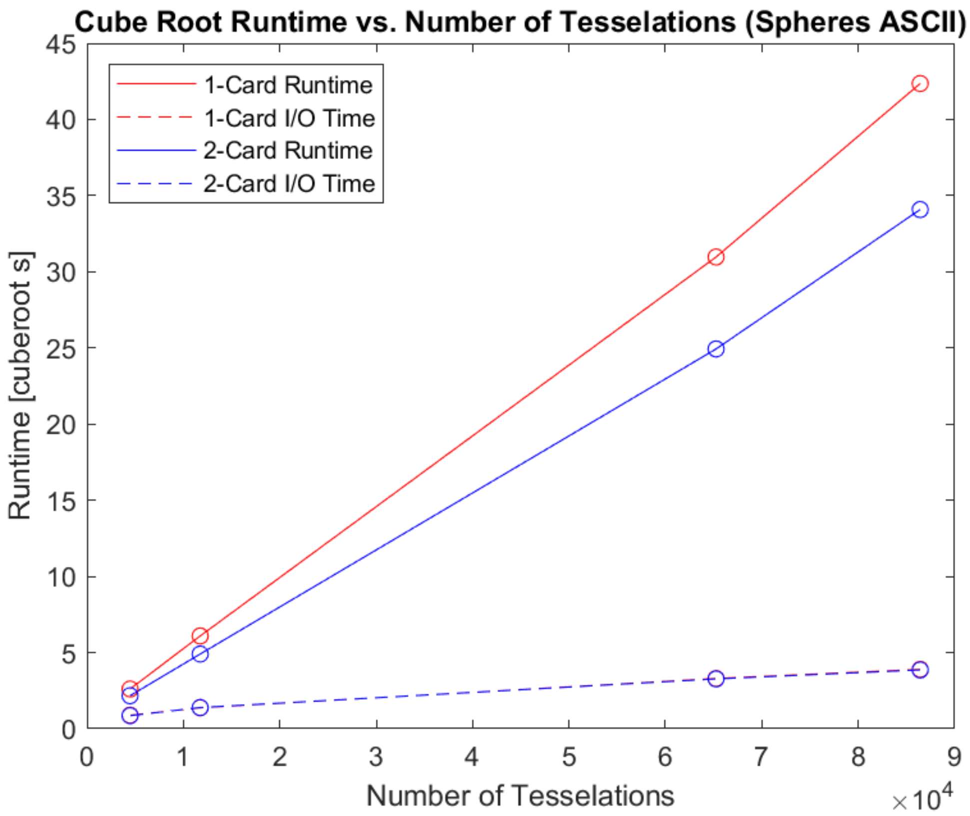
Binary STL's load much faster than ASCII





Discussion:

Due to three-loop nature of self-intersecting code, $O(n^3)$ asymptotic runtime is expected

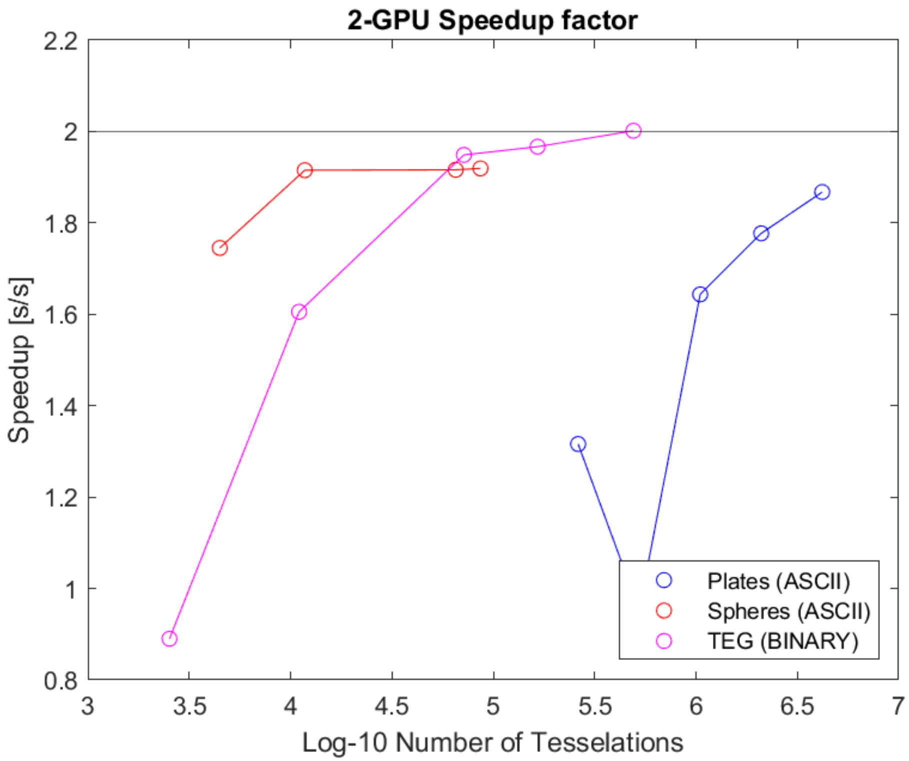


Discussion:

Per case, the speedup factor approaches 2x

Speedup hampered by I/O time, which cannot be parallelized, and grows linearly with problem size

Using binary STL files greatly improves on I/O time, improving speedup



Conclusion:

View factor computation lends well to parallelized computing on GPU

Given fast I/O, the speedup factor for 2 cards can approach 2x for large problems, without altering results

GPU vastly outperforms CPU in runtime, and the CUDA implementation is faster than a similar OpenCL implementation applied to the same problems
(Hancock et. al)

Given the need for fast and accurate view-factor calculations in heat transfer simulation, a need is presented for view-factor solvers that offer flexible options for backend computation
(e.g. multicore CPU or GPU)

Worth investigating single vs. double-precision GPU compute