Fast and Robust External Solar Shading Calculations using the Pixel Counting Algorithm with Transparency

Joel Hoover, Timur Dogan
jah552@cornell.edu, tkdogan@cornell.edu
Environmental Systems Lab, Cornell, Ithaca, New York, USA

Abstract

External solar shading calculations play an important role in energy models for buildings. Current simulation software implements polygon clipping algorithms for these calculations. However, polygon clipping suffers from several limitations, such as high computational costs and, for complex geometry, robustness issues. These weaknesses are a major bottleneck for the simulation of large scale urban building energy modelling (UBEM). This paper introduces a graphics hardware accelerated shading algorithm that uses pixel counting and supports transparency. The algorithm has been integrated in EnergyPlus and proves to be on average 2 times faster than EnergyPlus's shading algorithm, while maintaining an accuracy of 0.01.

Introduction

The current trend of population-growth and urbanization requires construction and densification of urban centers globally. With 33% of carbon emissions coming from buildings, this development is worrisome, but also presents a great opportunity to mitigate climate change through effective use of solar energy and energy efficiency measures in buildings. One crucial component of sustainable and passive environmental performance driven design is solar control and effective use of the suns energy in order to reduce heating and cooling loads by utilizing passive solar gains in the winter and effective solar shading in the summer (Olgyay, Olgyay, and others 1976).

Recent developments in the field of dynamic building simulation have provided holistic, well-validated building energy simulation frameworks such as EnergyPlus (Crawley et al. 2001), TRNSYS (Klein 1979) and others (Crawley et al. 2008) that can predict most relevant environmental performance indicators. Such software allows designers to optimize a buildings orientation and solar exposure to minimize HVAC energy consumption. This optimization is even more important at the urban scale – where building volume, position and orientation are determined – and the impact of these design decisions affect many buildings.

Since the solar environment greatly impacts a building's energy performance, reliable solar shading calculation is one of the most important components of these tools. Solar shading consists of three components: direct beam radiation, indirect diffuse radiation, and reflected radiation. In this paper, we are only concerned with beam radiation. The equation for calculating the solar heat gain from direct beam radiation is

$$Q_{\mathrm{beam}} = \alpha I_{\mathrm{b}} \frac{A_{\mathrm{s}}}{A_{\mathrm{t}}} \cos\theta \qquad (1)$$

where $Q_{\mathrm{beam}} =$ solar heat gain per unit area, $\alpha =$ fraction of solar absorption of the surface, $I_{\mathrm{b}} =$ intensity of direct beam radiation, $A_{\mathrm{s}} =$ sunlit surface area, $A_{\mathrm{t}} =$ total surface area, and $\theta =$ angle between sun's rays and the surface's normal vector (EnergyPlus Development Team 2016). The most computationally intense part of this equation comes from the A_{s} component, which requires consideration of the surface's context and determining what amount of the surface is exposed to the sun and not hidden in the shadow of another surface. Rather than calculate A_{s} directly, the term $(A_{\mathrm{s}}/A_{\mathrm{t}})\cos\theta$ is often calculated instead. This value is called the projected sunlit surface fraction (PSSF), and our focus is to efficiently calculate this value.

There are two general approaches to finding the PSSF at any time during the simulation. The EnergyPlus method is to calculate the PSSF for each surface of the building for each simulation time step throughout the entire year. The other method, employed by TRNSYS, is to divide the sky into many small sky patches, and calculate the PSSF of each surface as if the sun was in the center of each sky patch. Then, to get the PSSF for any time step, the PSSF of the sky patches around the sun position are averaged. There are advantages and disadvantages to both approaches, but both require an underlying algorithm to calculate the PSSF values for the sun at a given location in the sky.

There are four classes of algorithms that can be used to calculate PSSF values: Trigonometric algorithms for awnings, fins, and horizons; ray tracing; polygon clipping; and pixel counting.

First, trigonometric algorithms allow for the PSSF to be calculated quickly and exactly, but only work for very simple shading devices, such as overhangs and wing walls (McCluney 1990). While these algorithms are still implemented in modern simulation software, namely TRNSYS, their strict requirements on the shading geometry limit their application. For complicated shading systems or for contextual shading, such as for buildings on the opposite side of the street, another shading algorithm must be used.

Second, polygon clipping algorithms is a class of geometric algorithms that calculate the PSSF accurately, and unlike trigonometric algorithms, they do not place overly strict restrictions on the geometry. As such, they have been implemented and are still widely used for calculating PSSF. The general outline for a polygon clipping algorithm is as follows: First, take all the surface and all the shading geometry in the scene, and project them onto the plane perpendicular to the direction of the direct beam radiation. Then, for each shading surface in the scene, the projected polygon is "clipped" from the projected polygon of the surface. Finally, take the area of the resulting polygon and divide it by the area original surface to get the PSSF. For a more comprehensive description of the various polygon clipping algorithms, refer to (Hiller, Beckman, and Mitchell 1996). Polygon clipping has many limitations, however, which makes it less than ideal. It still places some restrictions on the geometry: for most algorithms, polygons must be convex, and robustness issues can occur when multiple shadows overlap, which can cause the algorithm to fail even for relatively simple geometry. Further, current implementations are extremely slow, with the polygon clipping shading algorithm being one of the most time-consuming component of the simulation.

Third, ray-tracing based methods directly simulate individual rays of sunlight falling on the surfaces bouncing around the scene. Because the simulated rays are independent and can be reflected, ray-tracing can simulate not just direct beam radiation, but also reflected radiation. Therefore, it is a promising approach to improve accuracy of both direct and diffuse calculation methods, as well as distribution of sunlight on the interior of the building. However, since these methods need to simulate a large number of independent rays to get accurate results, they end up being computationally expensive and are therefore less suitable for urban/early design applications. EnergyPlus calculates that the diffuse component using a coarse ray casting method, as opposed to full ray tracing, and assumes that this is sufficiently accurate. Hence, the computational overhead required for ray-tracing is only justifiable in special cases where propagation of the diffuse radiation plays a significant role.

Finally, pixel counting algorithms allow for fast approximation of the PSSF through computer graphics based methods: they approximate the PSSF by rendering the scene to an image buffer and counting the number of pixels visible from the suns viewing angle to estimate the exposed area of each surface (Yezioro and Shaviv 1994). The most recent implementation of such an algorithm is described by Jones et al. (Jones, Greenberg, and Pratt 2012) and it is shown that shading calculations can be accelerated significantly without sacrificing accuracy. The pixel counting approach can handle complex geometry, including elaborate shading devices, without a loss in accuracy, which alleviates the current necessity to oversimplify the energy model's input geometry. Jones et al.'s implementation is unfortunately not publicly available. Another limitation of their implementation is the lack of transparency support. Transparency is relevant in early building and urban design to model vegetation coverage, which would need prohibitively complex geometry to model exactly, but can be approximated by only a few transparent polygons. Vegetation also often has seasonal changes, which can easily be modelled by assigning a schedule to the transparency. Additionally, shading systems that are either change dynamically or include fritted or metal mesh facades can be modeled using transparency, which is useful for efficiency purposes. In urban design, this level of detail is generally not modelled geometrically due to CAD software limitations and the large burden on the designer. Using simple surfaces with transparency keeps the polygon count manageable, while still providing a close approximate of more detailed designs. Hence, this paper describes a publicly available solar shading simulation tool that is GPU accelerated, supports transparency, and can be used in conjunction with EnergyPlus.

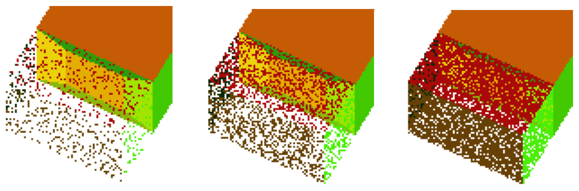

Figure 1: Capture of the pixel counter's internal color buffer for Case #8 at 15% (left), 40% (middle), and 80% (right) transparency.

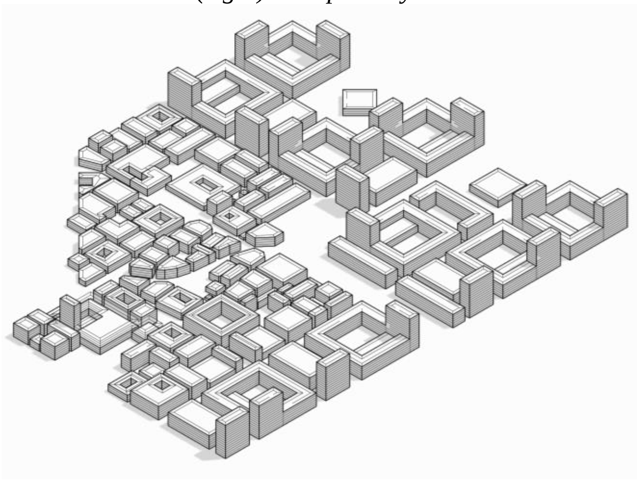

Figure 2: Urban Test Case #9

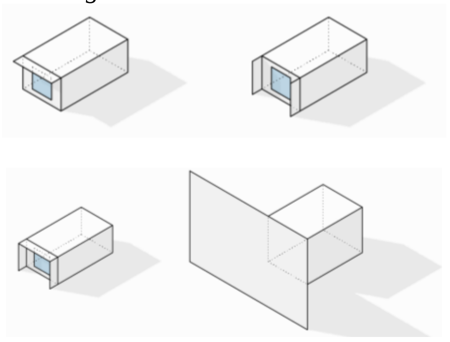

Figure 3: ISO Validation Cases #1 (left) through #4 (right).

Methods and Implementation

The Pixel Counting Algorithm

Our pixel counting algorithm for calculating PSSF is inspired by the algorithm proposed by Jones et al. The general outline of this algorithms is as follows: first, the entire scene, except for the surface of interest, is rendered (drawn) to an image buffer. Then, an OpenGL query object is generated, and the surface of interest is rendered to the buffer. (It is important to note that depth information is retained as part of the image buffer, so that anything in the scene that is in front of the surface will occlude the surface at the pixels of overlap, even though the surface is rendered later.) Next, the query object is read, yielding the number of pixels in the buffer that were affected by the surface of interest. Finally, the pixel count is converted into area, from which the PSSF can be calculated. This entire process is then repeated for every surface of the building. The renders are performed with a special setup: the camera is configured so that the projection is orthogonal (to model the sun's rays as being parallel), that the angle of the rays correspond to the sun's position in the sky, and that the surface of interest takes up as much of the render buffer as possible. Finally, each surface is given a unique color as an identifier.

For our algorithm, we make three major modifications. First, instead of scaling the render so that the surface of interest covers the entire buffer, so that each surface has its own transformation, the render is scaled such that all surfaces are visible with only one transformation. This allows the entire scene to only be rendered once, rather than needing the entire scene to be rendered again and again for every surface. To find the number of pixels in the final buffer that belong to a given surface, we activate an OpenGL query object, draw the given surface again, and then read back the result of this query object. These queries are performed for every active surface in the scene. Because we need all the active surfaces to be contained in the final render, we can no longer perform the per surface scaling à la Jones et al.'s algorithm. However, our method greatly reduces the number of renders required, as we only need one per sun position rather than one per active surface per sun position.

Our second modification is to add probabilistic model of transparency as follows: we apply a shader that will pseudo-randomly discard pixels from triangles as they are being rendered, where the probability of a pixel being dropped is equal to one minus the opacity of the pixel. That is, a triangle with opacity 1.0 will have no pixels discarded (fully opaque), a triangle with opacity 0.0 will have all pixels discarded (fully transparent), and a triangle with opacity 0.5 will have approximately half of its pixels discarded (50% transparent). The pseudo-random algorithm in the shader is seeded with the id of the surface, so that same surface rendered twice will have the exact same pixels discarded. A visualization of the transparency in action can be seen in Figure 1, which are captures from the pixel counter for test Case #8.

For our final modification, we add a stencil buffer to handle intersecting surfaces. Since we place no restrictions on the geometry, it is possible for two surfaces to intersect each other, and the pixels at the intersection will be counted twice. To resolve this issue, we add a stencil buffer to our render target; then during the second query pass, drawing to a pixel sets the stencil buffer at that pixel to 1 and cause further draws to that pixel to be discarded. Thus, each pixel will only be counted once.

Implementation

We have implemented our algorithm in C# using the OpenTK bindings to OpenGL. It requires OpenGL 2.0 or higher, and the extension EXT_Framebuffer_Object must be supported. We have also designed several Grasshopper components that allow our algorithm to be used in the Rhino CAD software. Finally, we have a modified version of EnergyPlus that reads in the shaded fractions from a file, allowing us to perform energy simulations using our pixel counting generated values. The implementation is available at es.aap.cornell.edu. The pixel counting codebase is split into several sub-projects: ShadingBase, PixelCountingLib, PixelCountingTest, and PixelCountingPlugin. ShadingBase provides as set of classes that abstract away certain details of the pixel counting code, such as geometry specification and error reporting. This would allow, for example, alternate pixel counting algorithm, or even a polygon clipping algorithm, to be implemented with the same interface, allowing easy comparison between the two algorithms. PixelCountingLib is the actual implementation of our pixel counting algorithm, and implements the interface given in ShadingBase. PixelCountingTest contains a few test cases for the PixelCountingLib implementation. Finally, PixelCountingPlugin provides the Grasshopper components that allow our algorithm to be used in Rhino.

To control the number of pixels used for the render buffer, our implementation calculates the optimal size given a "resolution" from the user. A resolution is given in area (in cm^2), and defines the maximum area that a pixel can have. Once the geometry is given, the implementation calculates the number of pixels needed to render the geometry and then uses that number of pixels internally.

Testing Methodology

We test our implementation against 9 test cases, with each case evaluated for accuracy, speed, and robustness. Accuracy is validated against analytic values (if available), and/or against the shaded fractions generated by EnergyPlus. Each test case is evaluated at the resolution sizes of $4000cm^2$, $400cm^2$, $40cm^2$, and $4cm^2$.

Test cases #1-#6 are taken form an ISO standard for shading calculation validations ("ISO 13791:2012-03 Thermal Performance of Buildings - Calculation of Internal Temperatures of a Room in Summer without Mechanical Cooling - General Criteria and Validation Procedures" 2012). Test cases #1-#4 are illustrated in Figure 3. We validate our implementation against the values for the sunlit fraction at various sun positions as given in the standard. We additionally validate and speed test our implementation against EnergyPlus by comparing the PSSF values for every one hour time step in a year simulation at Phoenix, Arizona and at Anchorage, Alaska, for each case. Additionally, both validation tests are done at the resolutions of 4000cm^2, 400cm^2, 40cm^2, and 4cm^2.

Test case #7 is a sequence of small cylinders outside of the standard south facing building in test case #1-#3. There are 75 cylinders, each 4m long and with a 20cm diameter. They are equally spaced vertically to take up the entire 3m height of the building, and are placed 0.1m away from the south wall. We then select the sun positions that are due south at elevation 0º, 5º, and 10º elevation and use pixel counting at various resolutions to find the PSSF. Additionally, a timing test is performed for the pixel counting implementation to calculate the PSSF at 4000 random sun positions. This test will allow us to determine how the pixel counting algorithm handles extremely fine geometry, what resolution is needed to attain satisfactory accuracy, and how well the implementation handles a large number of shading surfaces.

Test case #8 is again the standard south facing building and window from cases #1-#3, but we add a transparent enclosure that extends 2m from the south-facing wall. We then run an EnergyPlus simulation at Phoenix, Arizona and at Anchorage, Alaska, and use the hourly PSSF values to validate our transparency implementation. The enclosure is tested at 0%, 15%, 40%, and 80% transparency (where 0% is fully opaque and 100% fully transparent), and the resolution for our pixel counter is tested at 4000cm^2, 400cm^2, 40cm^2, and 4cm^2. This test will compare the accuracy of our transparency implementation to the EnergyPlus transparency algorithm. Test case #9 is a full scale urban model containing 121 buildings in a 600m by 800m block. The model is illustrated in Figure 2. Windows are placed on each wall of all buildings to cover 95% of the width and 95% of the height of the wall. We then perform a yearly EnergyPlus simulation at Phoenix, Arizona and at Anchorage, Alaska and compare the resulting PSSF against our pixel counting implementation at resolution 4000cm^2, 400cm^2, 40cm^2, and 4cm^2. This test will function as a stress test for our implementation, testing if it is robust enough to handle an entire urban scene with many active and shading surfaces.

All pixel counting tests and EnergyPlus simulations were run on a Mid-2014 MacBook Pro with 2.6 GHz Intel Core i5 with 8GB RAM with an Intel Iris 1536 MB. The only exception is Case #9, where the EnergyPlus simulations were performed on a Windows 10 Desktop with a 3.0 GHz Intel i7 6950x and 64 GB RAM. The version of EnergyPlus used is 8.5.0.

Results

The results from the DIN EN ISO 13791:2012-08 validation are shown in Figure 4. Additionally, the acceptable error range as specified by the standard is highlighted in green in the graphs. In graphs (a) and (b), we see that the errors at 4000cm^2 and 400cm^2 are well outside the acceptable tolerances and clearly fail validation. The 40cm^2 resolution in graph (c) almost passes, but fails at the 7:30 time step and Case #6 at noon, and Case #2 pushes the bounds from 8:30 to 9:30. Finally, at the 4cm^2 resolution in graph (d), all errors are well within the tolerance, except again for at 7:30 and Case #6 at noon.

While these results seem to indicate that none of the resolutions tested are accurate enough, we note that all the significant errors at 4cm^2 and 40cm^2 occur while the sun is at low angle relative to the window. At such low sun angles, very little radiation is hitting the window, and so large errors in the sunlit fraction of the window result in relatively small errors in the amount of radiation received. For example, the south-facing windows in Cases #1-#4 would receive only 7.7% of the radiation at 7:30 than at noon, and so a 4% error in sunlit fraction that would be within tolerance at noon is equivalent to an almost 50% error in sunlit fraction at 7:30. Likewise, for Case #6 at noon, the sun position is in the plane of the window, so no radiation will fall on the window, and so any value given for sunlit fraction is meaningless, as it will be multiplied by 0 before used for any further calculations. Thus, the large apparent error in Case #6 at noon is meaningless.

Because of these issues with sun position at low angles, we use PSSF instead of sunlit fraction in our further accuracy evaluations, as it is directly proportional to the amount of radiation hitting a surface.

The results of the validation of Cases #1-#6 against EnergyPlus are presented in histograms in Figure 5 (a)-(d), and the mean and standard deviation in the errors are reported in Table 1. The times in which the PSSF is zero have been excluded. We see that the distribution of errors roughly follows a normal distribution and is centered around 0. At 40cm^2 resolution, all the errors are within 0.05, and the 4cm^2 resolution, within 0.01. However, the Anchorage location has consistently higher error than the Phoenix location. Figure 6, which plots error against sun elevation, explains why: both the Phoenix and Anchorage locations have very similar error distributions at a given sun elevation, and that lower sun elevations has tends to have larger error than higher sun elevations. Since Anchorage has lower sun elevations than Phoenix, Anchorage has a greater average error in the PSSF.

The timings for Cases #1-#6 are presented in Table 2. To generate the EnergyPlus timing, two simulations were run for each case, once by running EnergyPlus by recalculating shading every day, and one by recalculating shading one a year, and then the difference in timings between the two runs is reported. The EnergyPlus shading calculations took between 1.82 and 2.12 seconds. By contrast, the pixel counting simulations took between 0.68 and 0.89, except for the 4cm^2 resolution for Cases #4 and #6, which took 1.4 and 1.5 seconds, respectively.

For small cylinder Case #7, a simulation using EnergyPlus was attempted. However, due to the extremely high polygon count of the cylinders, EnergyPlus did not advance past the "Initializing Simulation" phase in over 24 hours, so the simulation was aborted. Luckily, there is an approximate analytic solution to this shading problem if we limit the sun position to be due south with an angle of elevation θ, from the horizon at $\theta = 0°$ to the elevation where the cylinders block all

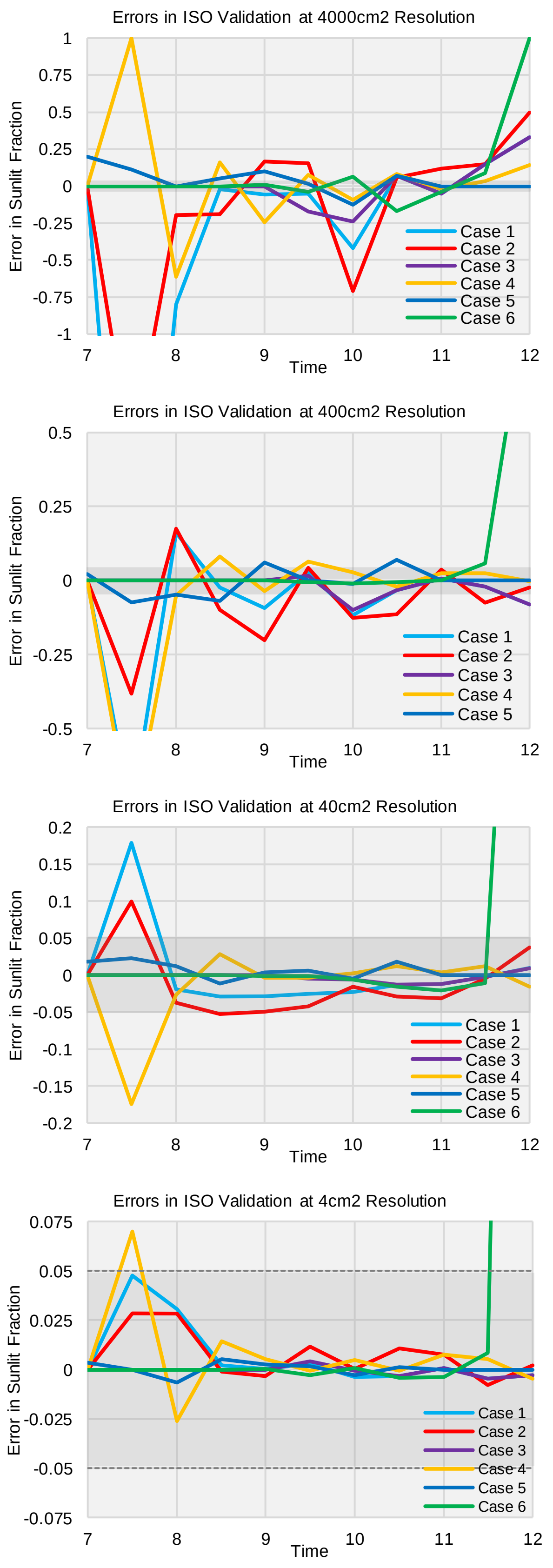

Figure 4: The error in the ISO validation test at each sun position during the sample day at resolutions (a) 4000cm², (b) 400cm², (c) 40cm², and (d) 4cm².

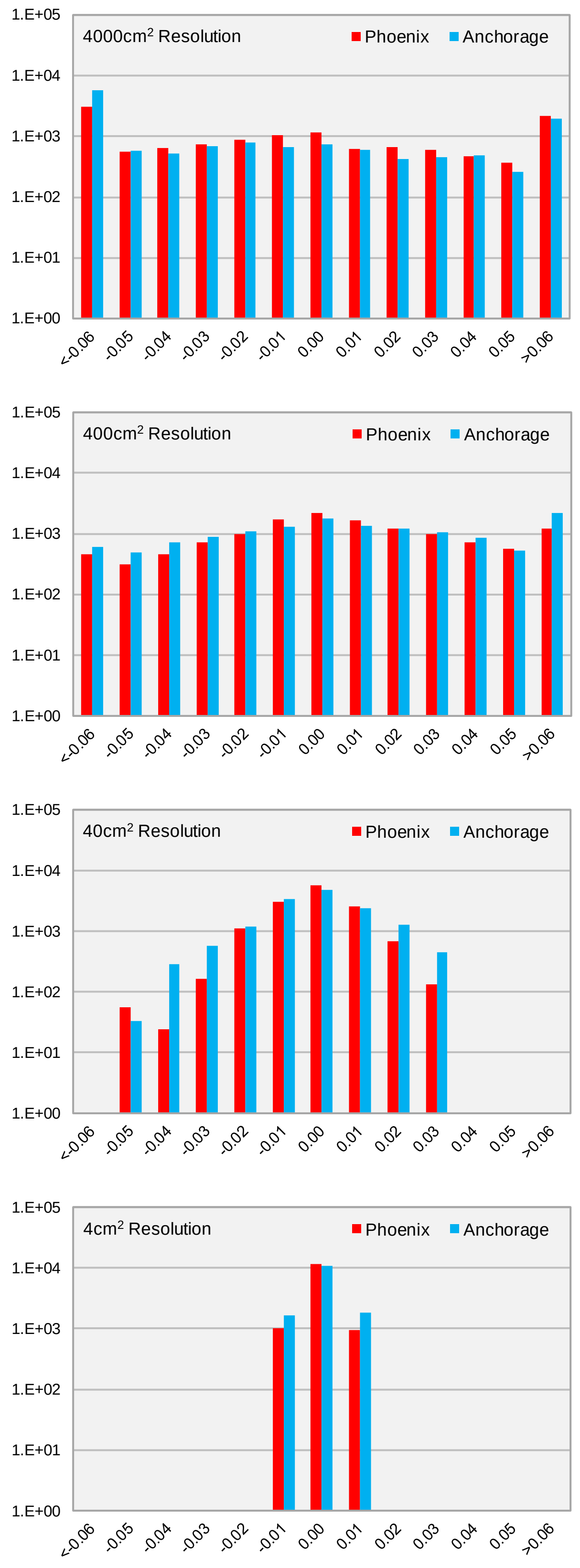

Figure 5: Histograms of the error distribution against EnergyPlus values for all Cases #1-#6 at Phoenix and Anchorage simulations at resolutions (a) 4000cm², (b) 400cm², (c) 40cm², and (d) 4cm².

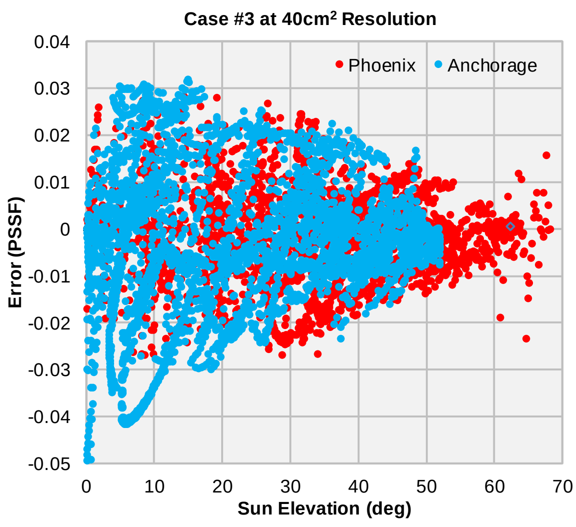

Figure 6: Plot of error in PSSF vs. sun elevation for ISO Case #3 at 40cm^2 resolution at Phoenix and Anchorage.

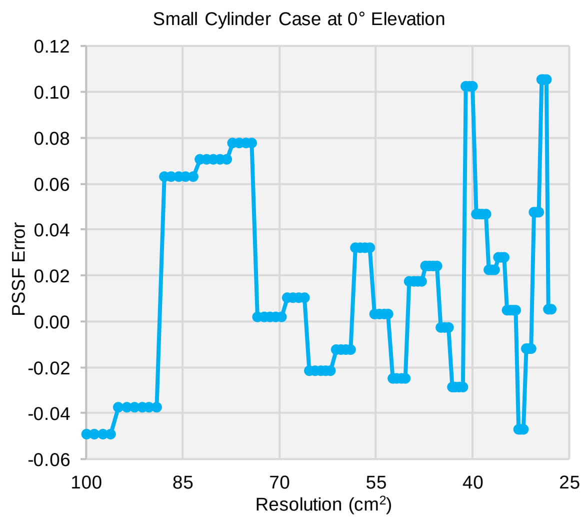

Figure 7: Plot of error in PSSF vs. resolution for small cylinder Case #7 at 0º elevation.

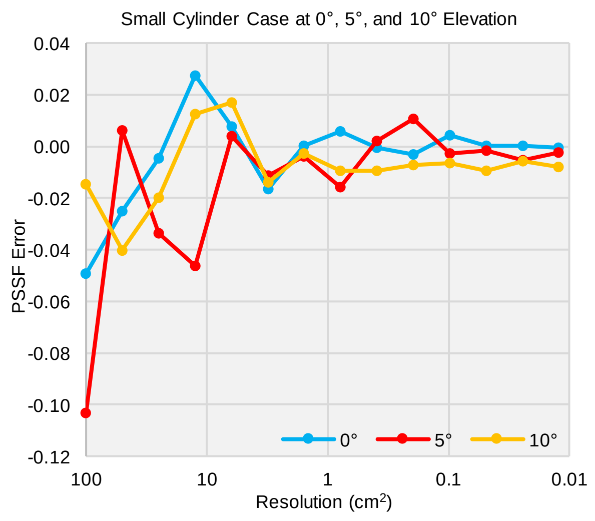

Figure 8: Plot of error in PSSF vs. resolution for small cylinder Case #7 at 0º, 5º, and 10º elevations over resolutions 100cm^2 to 0.01cm^2.

Table 1: EnergyPlus validation errors for Cases #1-#6.

Place	Resolution	Mean	Std. Dev.
Phoenix	4000 cm^2	-0.0144	0.0970
	400 cm^2	0.0058	0.0363
	40 cm^2	-0.0014	0.0109
	4 cm^2	-0.0001	0.0033
Anchorage	4000 cm^2	-0.0496	0.1211
	400 cm^2	0.0109	0.0464
	40 cm^2	-0.0018	0.0142
	4 cm^2	0.0000	0.0041

Table 2: The timing results of ISO Test Cases #1-#6 at Phoenix for EnergyPlus shading calculations and pixel counting at various resolutions.

Case	Energy Plus	Pixel Counting			
		4000cm^2	400cm^2	40cm^2	4cm^2
#1	1.90	0.70	0.69	0.76	0.80
#2	2.06	0.68	0.74	0.74	0.72
#3	1.82	0.72	0.68	0.89	0.78
#4	2.12	0.71	0.69	0.76	1.40
#5	1.98	0.73	0.73	0.67	0.82
#6	1.82	0.70	0.72	0.72	1.50

Table 3: The accuracy statistics for Case #8 at various resolutions.

Resolution	Transparency	Mean	Std. Dev
4000 cm2	0%	0.0084	0.1395
	15%	0.0062	0.1355
	40%	0.0051	0.1207
	80%	-0.0024	0.0777
400 cm2	0%	0.0050	0.0446
	15%	0.0049	0.0434
	40%	0.0029	0.0400
	80%	-0.0001	0.0253
40 cm2	0%	-0.0015	0.0138
	15%	-0.0007	0.0136
	40%	-0.0010	0.0129
	80%	-0.0004	0.0082
4 cm2	0%	-0.0001	0.0044
	15%	0.0003	0.0044
	40%	-0.0001	0.0040
	80%	0.0000	0.0026

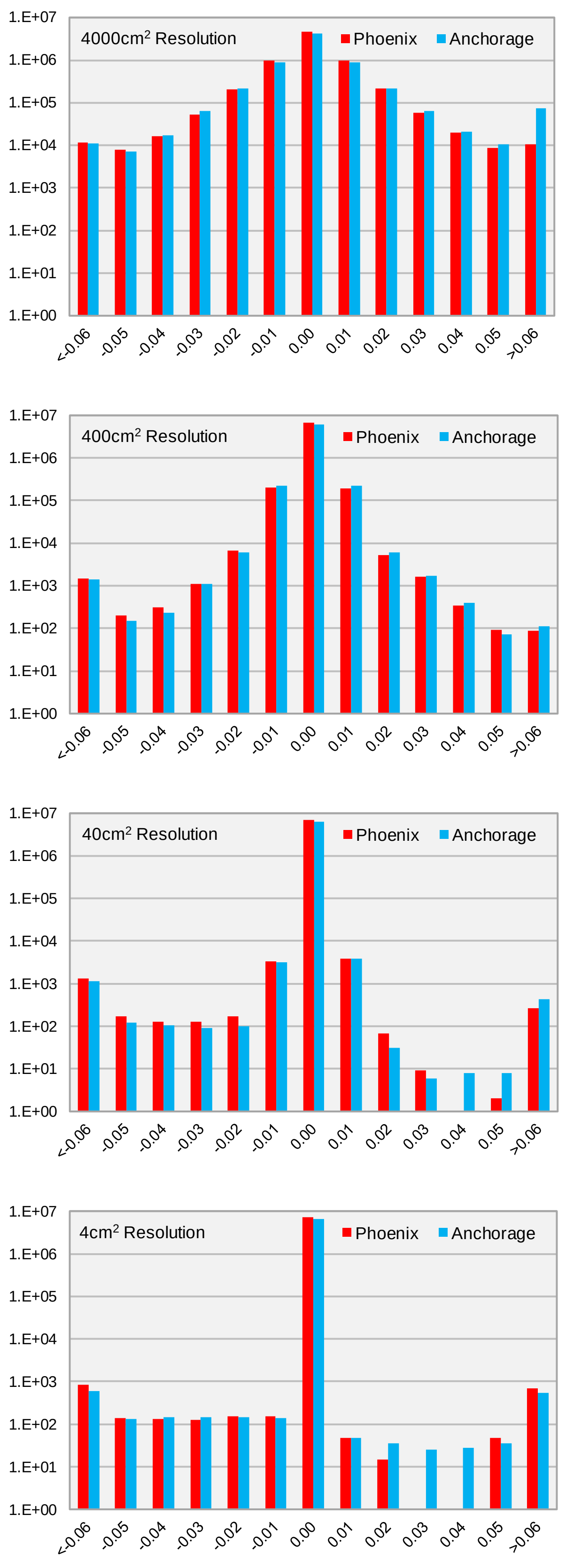

Figure 9: Histograms of the error distribution against EnergyPlus values for all Cases #1-#6 at Phoenix and Anchorage simulations at resolutions (a) 4000cm^2, (b) 400cm^2, (c) 40cm^2, and (d) 4cm^2

Table 4: The timing results of Urban Case #9 for EnergyPlus shading calculations and pixel counting at various resolutions.

		Piecewise Model	Full Model
EnergyPlus		14758 s	—
Pixel Counting	4000 cm^2	1789 s	852 s
	400 cm^2	1975 s	1020 s
	179 cm^2	—	1218 s
	40 cm^2	3816 s	—
	4 cm^2	20930 s	—

Table 5: Statistics on errors for Cases #9.

Place	Resolution	Mean	Std. Dev.
Phoenix	4000 cm^2	0.00005	0.0257
	400 cm^2	-0.00002	0.0105
	40 cm^2	-0.00001	0.0057
	4 cm^2	0.00000	0.0025
Anchorage	4000 cm^2	0.00411	0.0595
	400 cm^2	-0.00002	0.0118
	40 cm^2	0.00000	0.0065
	4 cm^2	0.00000	0.0022

light at $\theta = 60°$. This solution is presented in Equation 2 below.

$$PSSF = \cos\theta - 1/2 \qquad (2)$$

This equation is, however, only an approximation, as it does not account for cylinders near the edge of the window that only part of the cylinder shades the window. The maximum error is the proportion of the diameter of the cylinder compared to the window height, which is $20cm/2m = 0.01$.

Figure 7 presents the error vs resolution for the sun at 0° elevation for 100 resolutions between 100 cm^2 and 27.8 cm^2. Within these resolutions, the error fluctuates between -0.049 and 0.105. Figure 8 presents the error vs resolution for the sun at 0°, 5°, and 10° at 14 resolutions between 100 cm^2 and 0.012 cm^2. At the coarse resolutions, the errors are as large as -0.103, but as the resolution increases, the error decreases down to errors smaller than 0.01 for the smallest resolutions between 0.1 cm^2 and 0.012 cm^2. Together, these tests show that small changes in resolution can lead to large changes in error when the resolution is too coarse for the geometry, but as the resolution increases, this variation decreases, and the error can be bounded. Finally, for the timings test of Case #7, it took the implementation only 37.6 seconds to perform the 4000 renders, which is extremely good performance, as the cylinders are constructed of about 1.6 million triangles.

For Case #8, Table 3 presents the mean and standard deviation for the errors against the PSSF reported by

EnergyPlus. We again see that the means are very close to zero, and that the standard deviation decreases as resolution increases. One interesting observation is that the standard deviation is smaller for more opaque surfaces, although this is likely due to the PSSF itself being smaller, causing errors to naturally shrink as well.

Finally, we have Case #9. We had many issues attempting to simulate the large urban model with EnergyPlus. First, we attempted to run a simulation of the entire model, but this simply caused EnergyPlus to crash. Next, we generated a piecewise model, which simulated each building individually, including as shading surfaces any building within 150 meters. While a few of these simulations took under 10 minutes, a majority took over an hour, with some taking as long as 23 hours. In total, the simulations took 55 days, 20 hours, and 21 minutes of CPU time. In addition to the extremely long runtime, many of the simulations reported “severe errors” during the shading calculations, and that the “Shadowing values may be inaccurate.” So, we performed one further simplification to the piecewise model: only selecting as shading surfaces walls that are within 150 meters and face the center of the simulation building. This finally yielded simulations that completed in a reasonable amount of time, 19 seconds in the fastest case and 282 seconds in the longest case. However, there was still one simulation that reported severe errors during the shading calculations, and 20 other files that, while not reported as severe, produced warnings during the shading calculation that indicate an error occurred while generating some of the shaded fractions.

As opposed to EnergyPlus, the pixel counting implementation had no issues with running a simulation on the entire model. However, due to the hardware only supporting up to 8192 pixel by 8192 pixel framebuffers, in addition to the huge 500 meter by 700 meter model, the pixel counter could only simulate up to 179 cm^2 resolution. We also ran the pixel counter against the piecewise model that EnergyPlus could simulate for the accuracy comparison. The timings for these are reported in Table 4.

Table 5 provides the statistics on error in the urban case for each resolution. As for the ISO Cases, we again see that the mean is extremely close to 0, and that the standard deviation is larger for coarser resolutions. Figure 9(a)-(d) provides histograms of the errors for each resolution using the same bins as in Figure 5. Note that these histograms are logarithmic on the vertical axis. The shape of the distribution is again normal like, but there also seems to be a slight left skew to the distribution, and it appears that, for large errors, there are more negative errors than positive. Furthermore, there are many errors at the extreme edges of the histogram. These errors, along with most of the moderate errors, originate from the 20 simulations that that produced the warnings during the shading calculations. If we exclude these simulations, the errors are much more tightly bound, especially for the finer resolutions.

Discussion

The results in the previous section are overall very positive. For example, for Cases #1-#6, our implementation can calculate the PSSF values with an error of less than 0.05 at 40 cm^2 and less than 0.01 at 4 cm^2 with an average time of under half that of EnergyPlus. We see a similar pattern for Case #9, the urban model, where over 99.9% of the errors are less that 0.01 at 40cm^2 resolution, and well over twice the speed compared to the EnergyPlus simulation. Finally, Case #7 shows that our pixel counting implementation can handle a large number of shading surfaces without stability issues or an astronomical increase in run time. There is also an interesting pattern in the timing of a model, where for relatively coarse resolutions, a change in the resolution have little impact on the timing, whereas it has a huge impact at higher resolutions. This is because there is a certain overhead with render calls and transfering counts from the GPU that is independent of the number of pixels in the render target. This is why, for example, nearly all of the ISO test cases have the same timings, except for Case #4 and #6, which, due to their larger geometry size, need significantly more pixels for the same resolution as the other ISO cases, which leads to a longer run time. The timings for the Case #9 are also rather interesting, as the full model ran about twice as fast as the piecewise model. This is because the piecewise model contains all of the same geometry as the full model, but forces it to be processed one building at a time. On the other hand, the full model batches all the geometry together into one large render, which allows for a faster runtime overall.

The results also show that errors in the pixel counting method are centered around 0. This is extremely important, as it means that our implementaion predicts the same total amount of radiation will enter the windows as EnergyPlus’s polygon clipping algorithm. Additionally, the tests consistantly show that increasing resolution does decrease the error, allowing a specific accuracty goal to be met. However, the tests also show that the accuracy is not just dependent on resolution, but also of the geometry being simulated. For example, the 0.01 error cutoff could be reached by the 40cm^2 resolution for the urban model, but required a resolution of 4cm^2 for Cases #1-#6 and a resolution of less than 0.1cm^2 for Case #7. In general, a finer resolution is needed to capture finer details in the geometry.

In modern architectural design culture, building performance simulations remain underutilized as “generative” design tools. Energy models tend to be especially underrepresented in the fast-paced early design phase. The importance of implementing evaluative tools during the early design phase, however, is self-evident given that decisions made at this point such as building proportions and their spatial interrelationship with the context, largely “make or break” the intrinsic energetic performance of a building. One reason for the lack of acceptance may be traced back to their slowness. The results presented in the previous section have shown that it is possible to significantly accelerate energy simulations and therefore may facilitate a wider adoption of energy modeling software in the earliest stages of architectural and urban design. Another, caveat of current energy modeling tools in the design environment is that they are often regarded as less sensitive to geometric changes. This perception is certainly related to the current geometric limitations that are imposed by the polygon clipping tools. The proposed pixel counting method in contrast can deal with complex high-polygon-count geometry, such as the cylinders in Case #7, in a more efficient manner and therefore allows designers to test complex geometries with ease.

Another benefit that is energy and daylighting studies can be conducted more consistently. While daylight models often directly utilize the architectural CAD geometry for the analysis, energy model geometry must be abstracted. While this is still true for geometry that is partaking in heat transfer processes, the proposed method would allow modelers to keep architectural CAD geometry for shading devices and context. This significantly facilitates the model generation for complex facade shading geometry but also in urban design applications.

Urban building energy modeling [UBEM] is a nascent field of research. Modelers that are interested in energy implications of cities with hundreds or thousands of building often rely on simplified models ranging from statistical methods to modeling archetypical buildings as dynamic BEMs and then extrapolating the results.
Speed and robustness achieve by implementing pixel-counting allows modelers to run multi zone building energy models within a feasible time. Simulations for the previously mention urban example included 121 buildings and completed within 1218 seconds, whereas EnergyPlus could not handle the model without extreme preprocessing.

Conclusion

In this paper, we proposed a new pixel counting-based algorithm with transparency support and performed accuracy validations and speed tests on an implementation of the algorithm in C# using the widespread OpenGL 2.0 technology. The results of the tests show that pixel counting is a viable replacement for polygon clipping: our implementation can calculate PSSF values with error less than 0.01 in less than half the time of EnergyPlus's polygon clipping implementation. Additionally, our implementation can handle very high detail shading devices composed of millions of polygons, as well as large urban models with relative ease. This is compared to EnergyPlus, which cannot at all handle such shading devices, and requires heavy preprocessing and simplification to be able to handle an urban scene. Finally, our algorithm, unlike Jones et al.'s, supports transparency, and thus has all of the features of EnergyPlus's current polygon clipping shading algorithm, and thus can function as a drop-in replacement. As such, we highly recommend that our implementation be incorperated into the main EnergyPlus codebase, and that other systems needing a direct radiation shading algorithm prefer a pixel counting-based approach rather than a polygon clipping-based approach.

Acknowledgement

The authors would like to thank the Cornell University David R. Atkinson Center for a Sustainable Future for funding this research as well as NVIDIA for supporting the project with a hardware grant.

References

Crawley, Drury B, Jon W Hand, Michaël Kummert, and Brent T Griffith. 2008. "Contrasting the Capabilities of Building Energy Performance Simulation Programs." *Building and Environment* 43 (4): 661–673.

Crawley, Drury B, Linda K Lawrie, Frederick C Winkelmann, Walter F Buhl, Y Joe Huang, Curtis O Pedersen, Richard K Strand, et al. 2001. "EnergyPlus: Creating a New-Generation Building Energy Simulation Program." *Energy and Buildings* 33 (4): 319–331.

EnergyPlus Development Team. 2016. "EnergyPlus Engineering Reference: The Reference to EnergyPlus Calculations." Lawrence Berkeley National Laboratory.

Hiller, Marion D.E., William A. Beckman, and John W. Mitchell. 1996. "TRNSHD — a Program for Shading and Insolation Calculations." University of Wisconsin-Madison.

"ISO 13791:2012-03 Thermal Performance of Buildings - Calculation of Internal Temperatures of a Room in Summer without Mechanical Cooling - General Criteria and Validation Procedures." 2012. Beuth.

Jones, Nathaniel L., Donald P. Greenberg, and Kevin B. Pratt. 2012. "Fast Computer Graphics Techniques for Calculating Direct Solar Radiation on Complex Building Surfaces." *Journal of Building Performance Simulation* 5 (5): 300–312. doi:10.1080/19401493.2011.582154.

Klein, Sanford A. 1979. TRNSYS, a Transient System Simulation Program. Solar Energy Labarataory, University of Wisconsin–Madison.

McCluney, R. 1990. "Awning Shading Algorithm Update." *ASHRAE Transactions* 96 (1): 34–38.

Olgyay, Aladar, Victor Olgyay, and others. 1976. *Solar Control & Shading Devices*. Princeton University Press.

Yezioro, Abraham, and Edna Shaviv. 1994. "Shading: A Design Tool for Analyzing Mutual Shading between Buildings." *Solar Energy* 52 (1): 27–37.