A Viewfactor Based Radiative Heat Transfer Model for Telluride

Austin Minnich
UC Berkeley

John Turner
LANL

Michael Hall
LANL

Outline

1. Where this fits into Telluride
2. Radiosity Ideas and Background
3. Viewfactors
4. The algorithm
5. Results from the algorithm
6. Future Work

Where this fits into Telluride

- Before the casting operation begins, the mold is heated.
- We need the final temperature distribution of the mold to pass to Telluride.

The final temperature distribution of the mold is given by:

$$\alpha \frac{\delta T}{\delta t} = q_{conv} + q_{cond} + q_{rad}$$

The conduction and convection modules are already in Telluride.

We need the radiation term, q_{rad}.

Currently, Telluride uses the boundary condition:

$$Q_i = A_i K \sigma (T_i^4 - T_{amb})$$

for the q_{rad} term.

K is the only factor for viewfactors, emissivity, etc.

- Since thermal radiation varies as T^4, radiative heat transfer is very important in determining the final temperature distribution of the mold.
- So we need a more accurate model to make sure the final temperature distribution is correct.

A more accurate model

This is what we have done this summer.

Our code takes into account:

- viewfactors
- occlusion

It does this using principles from *radiosity*.

Radiosity

Radiosity is often used in graphics to model scenes with diffuse surfaces.

It has one very simple idea:
The total energy leaving a face is equal to the sum of the emitted energy and the reflected energies.

Mathematically, this reads:

$$B_i = \sigma e T_i^4 + \rho_i \Sigma B_j F_{ij}$$

- ρ_i is the reflectance of face i
- B_i, B_j are the total energies from faces i, j
- F_{ij} is the *viewfactor* from face i to face j

If we rewrite the last equation like this:

$$\sigma e T^4 = B_i - \rho_i \Sigma B_j F_{ij}$$

We can turn the last expression into a system of equations:

$$\begin{bmatrix} 1 & -\rho F_{12} & -\rho F_{13} & ... & -\rho F_{1n} \\ -\rho F_{21} & 1 & -\rho F_{23} & ... & -\rho F_{2n} \\ -\rho F_{31} & -\rho F_{32} & 1 & ... & -\rho F_{3n} \\ -\rho F_{n1} & ... & ... & ... & 1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ ... \\ B_n \end{bmatrix} = \begin{bmatrix} \sigma e T_1^4 \\ \sigma e T_2^4 \\ \sigma e T_3^4 \\ ... \\ \sigma e T_n^4 \end{bmatrix}$$

This is the deceptively simple radiosity equation.

The Radiosity Equation

Properties of the Radiosity Equation

1. The solution of the radiosity equation is the total energy leaving each face.
2. It is guaranteed to converge by Gauss-Seidel or other iterative methods because it is diagonal dominant.
3. In order to generate this system of equations, we must calculate *viewfactors*.
4. Once we solve the system, we can easily determine the net radiant flux, q_{rad}.

Viewfactors

But first we must figure out the viewfactors.

Definition: *A viewfactor is the fraction of energy emitted from an area A_i in all directions directly intercepted by another area A_j.*

The formal mathematical definition of a viewfactor is:

$$dF_{ij} = \frac{dA_i dA_j \cos\theta_i \cos\theta_j}{\pi R_{ij}^2}$$

We evaluate this expression in its vector form:

$$dF_{ij} = \frac{A_i u_r A_j u_r}{\pi \|R_{ij}\|^2}$$

u_r is the unit vector of R_{ij}, the vector joining the two faces, and A_i and A_j are area vectors normal to the surface.

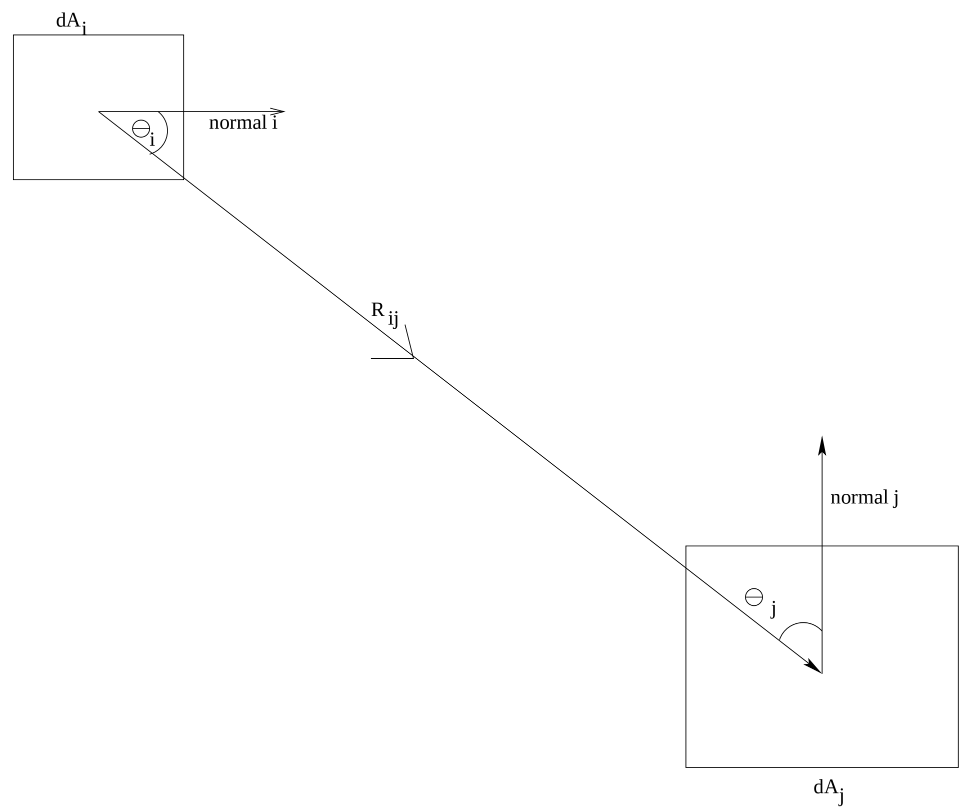

Figure 1: Example between two differential areas

Why viewfactors are important

- Viewfactors tell us how much energy from a face i actually goes to another face j.
- If these factors are calculated accurately, then we can be very accurate about how much energy goes from each face i to every other face j.
- This leads to a very accurate model for radiative heat transfer, meaning our net radiant flux matrix will be accurate.
- This makes our final temperature distribution more accurate.
- So viewfactors are essential to obtaining an accurate solution.

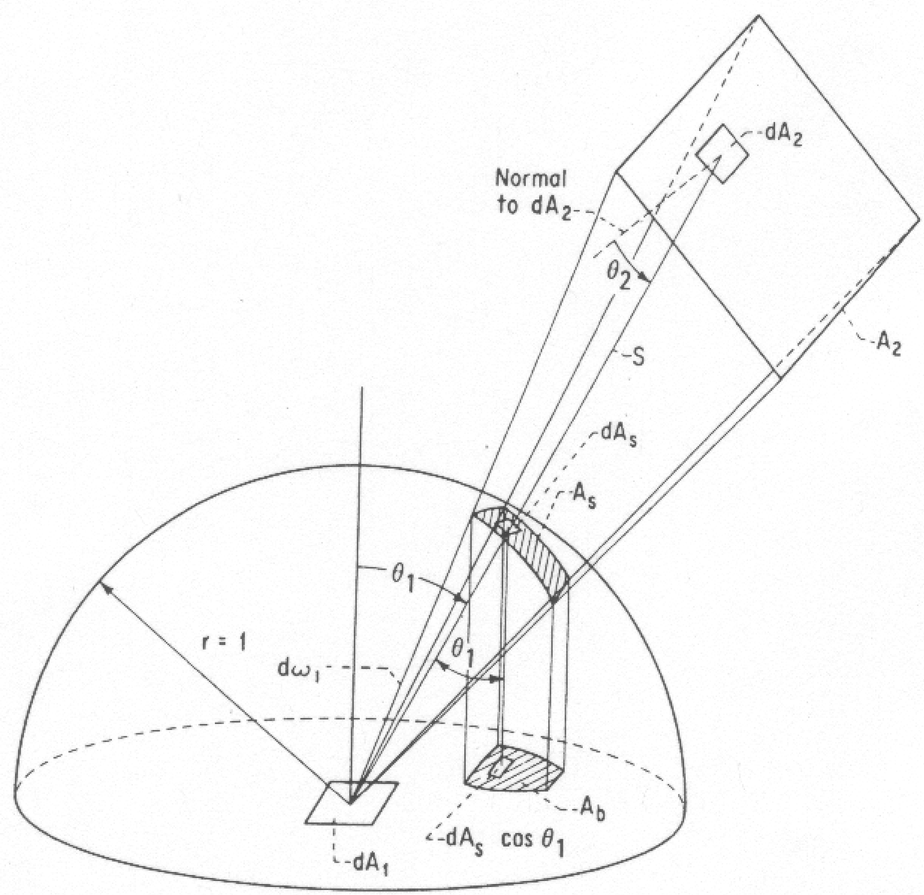

FIGURE 5-26 Geometry of unit-sphere method for obtaining configuration factors.

Figure 2: The Nusselt unit sphere

The algorithm

We need to form the viewfactor matrix so we can solve the system of equations.

But we can't just calculate the viewfactors for every face.

We have to make sure that the faces are visible to each other first.

The general form of the algorithm is:

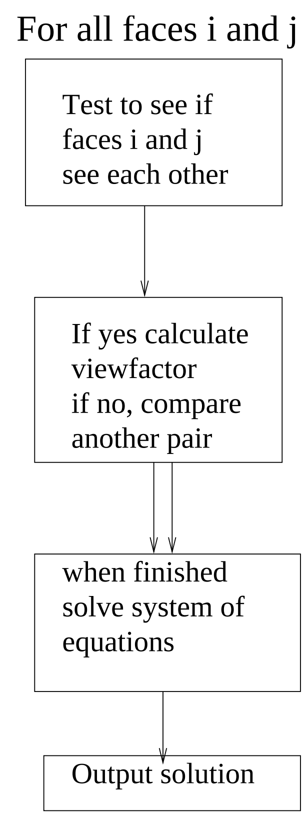

Figure 3: General flow chart for the algorithm

The Tests: The Dot Product Tests

First Dot Product Test: Check to see if the faces point in the same direction by taking the dot product of the two area vectors.

This would pass.

Figure 4: dot product is negative

This would not pass.

Figure 5: dot product is positive

The Second Dot Product Test

This test eliminates those faces pointing in opposite directions.

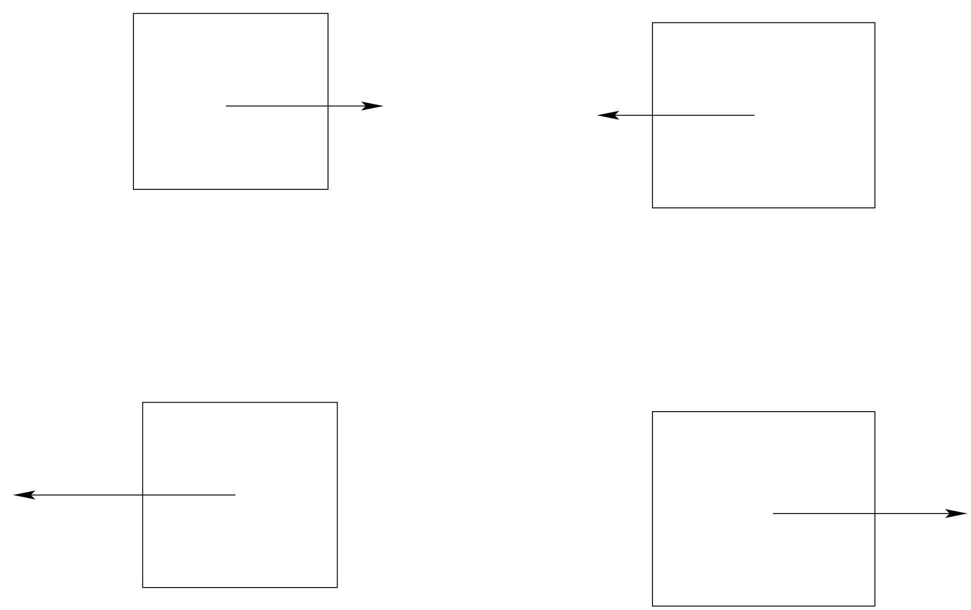

Figure 6: These two examples both have negative dot products, but only the top example is correct. This is solved by the second dot product test.

Occlusion

If the pair of faces pass the dot product tests, then we know they face each other.

But, we need to make sure there is nothing in the way before we can calculate the viewfactor.

In other words, we need to make sure, from the point of view of face i, that there is no face k in the way of face j.

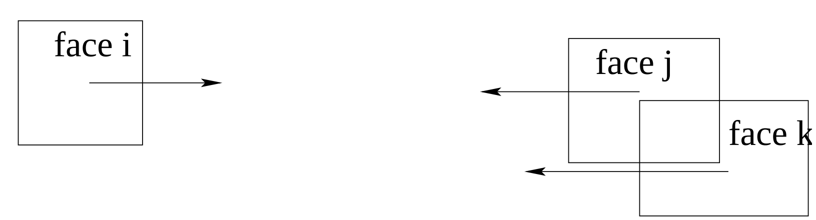

Figure 7: Example of a possible occlusion.

The Occlusion Routine

Occlusion is very hard to detect. Here is how the program does it.

For each face i

Rotate Coordinate System

For each face j

Project into xy plane

For each face k

Project into xy plane

Compare projections

If overlap: store index of overlapping fac
If no overlap: try another face

If none overlap: use face j in calculation
If overlap: use closest face in calculation

Figure 8: flow chart for the occlusion subroutine.

The Viewfactor Matrix

The viewfactor matrix is an n by n matrix which contains all the viewfactors from every face i to every other face j (n is the number of faces).

All of these tests help fill the viewfactor matrix.

- If a face fails any of these tests, then the viewfactor matrix at that index is set to 0.
- If a face passes all these tests then the viewfactor is calculated and stored in the matrix.
- If a face j is occluded by a face k, the viewfactor for face j is set to 0, and the viewfactor for face k is calculated and stored in the matrix.

Problems with our Occlusion Routine

Our routine does not handle partial occlusions.

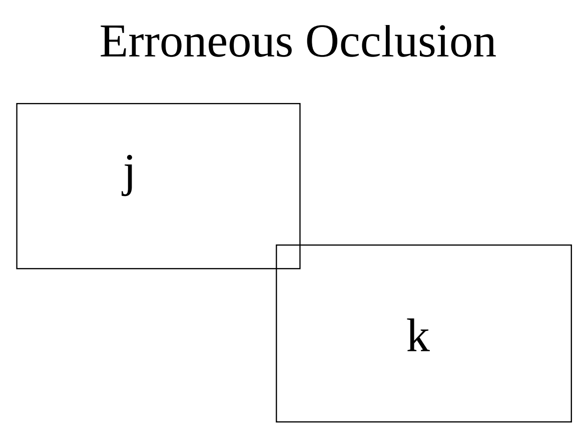

Figure 9: Our occlusion subroutine will register this as an occlusion even though face k occludes very little of face j.

Relating Radiosity and Net Radiant Flux

After we have solved the system of equations, we have a matrix with the radiosity of each face.

We need to get the net radiant flux and then the temperature(remember, we are still trying to get the final temperature distribution of the mold).

To get the net radiant flux, all we need to do is this:

$$q_{rad} = B_i - I_i$$

$$I_i = \Sigma B_j F_{ij}$$

Relating Net Radiant Flux and Temperature

Once we have the net radiant flux, we get temperature by:

$$\alpha \frac{\delta T}{\delta t} = q_{conv} + q_{cond} + q_{rad}$$

Using this program for q_{rad}, we can now get an accurate temperature distribution for the mold.

Results

To test the program, we ran the program on several spherical meshes and looked at how each sphere interacted with the others.

The following pictures are of *net radiant flux*, not of temperature. Thus an area that looks colder means that that area has less net radiant flux. This means that there is more incident radiation on the area.

Here are some pictures.

Future Work

So far we have worked on this code independent of Telluride. We now need to:

- Integrate the program into Telluride. John will probably do this after the workshop.
- Parallelize the code. This will also probably happen after the workshop.