Алгоритм реализации конструктивной блочной геометрии (CSG) на прямоугольных параллелепипедах, удаление невидимых линий и граней в невыпуклых многогранниках, подключение прозрачности при заливке цветом.

Редька Ал. В., Иванов К.А.

АО «НПП «Пульсар» г. Москва, Окружной проезд д.27.

Аннотация: В интерфейсе пакета программ [1], с использованием библиотеки OpenGL, реализована конструктивная блочная геометрия (CSG) на прямоугольных параллелепипедах, алгоритм удаления невидимых линий в каркасном представлении модели, заливка пользовательскими цветами, полупрозрачность.

Ключевые слова: Компьютерная графика, Z буфер, каркасная модель, CSG, полупрозрачность, OpenGL, рендеринг, препроцессинг, оптимизация рендеринга.

Введение

Во многих САПР (HFSS, Solid Work и др.) на этапе постановки задачи - создании компьютерной модели исследуемого устройства, а также на этапе визуализации полевых рассчитываемых величин, используются математические алгоритмы машинной графики. Они преследуют цель наиболее реалистично для пользователя изобразить моделируемое изделие и рассчитываемые поля физических величин. В данной статье будет рассказано о некоторых деталях программной реализации подобных алгоритмов в пакете программ [1] с использованием открытой графической библиотеки OpenGL [3], [5], многие функции которой реализованы аппаратно производителями графических ускорителей.

Предлагаемая в пакете программ [1] реализация алгоритмов компьютерной графики служит цели облегчения восприятия конечным пользователем моделей радиаторов, охлаждающих радиоэлектронное оборудование, на этапе постановки задачи, а также визуализации рассчитываемых тепловых полей на этапе анализа результатов вычисления. Кроме того, реализация подобных алгоритмов на ЭВМ представляет несомненный интерес, а результаты работы подобных алгоритмов визуализации вызывают положительные эстетические чувства.

Аналогичные, но более развитые алгоритмы визуализации, входят как составная часть многих САПР: ANSYS Icepak, ANSYS HFSS, CST Thermal and High Frequency simulations, Sonnet Suite, SolidWork.

1. Визуализация полупрозрачных объектов с использованием OpenGL.

Для создания полупрозрачных объектов используется смешивание. Ведь полупрозрачный объект не полностью закрывает собой сцену, находящуюся за ним, наблюдатель как будто смотрит через стекло или мутное стекло. Значение прозрачности α в этом случае отвечает за то, насколько через стекло хорошо видно. Самые ранние

разработки в этой области принадлежат Ньюэлу (Newell) август 1972 года [2]. Он предложил следующую формулу для результирующего цвета отображаемого пиксела:

$$resultColor = currentColor \times \alpha + fontColor \times (1-\alpha),$$

где curentColor - цвет отображаемого полупрозрачного тела без учёта прозрачности, fontColor - исходный цвет того, на что накладывается наш рисуемый прозрачный объект (цвет фона), α - интерполяционный коэффициент, лежащий в отрезке [0..1] - прозрачность, resultColor - результирующий цвет пиксела. Более детально теория полупрозрачности изложена в [2] стр. 410-416. Согласно спецификации [5] стр. 98-100 алгоритмическая поддержка формулы Ньюэла была в библиотеке OpenGL изначально уже в версии 1.0.

При создании [1] автор данной статьи воспользовался следующей последовательностью действий (про буфер глубины будет рассказано в п.2):

Алгоритм прорисовки полупрозрачных объектов в OpenGL [3]:

шаг 1. рисуются непрозрачные объекты в обычном режиме доступа

к буферу глубины (т.е. с предварительно заданным

glEnable(GL_DEPTH_TEST)).

шаг 2. включается смешивание glEnable(GL_BLEND);

шаг 3. устанавливается запрет на изменение значений в буфере глубины

glDepthMask(GL_FALSE);

шаг 4. рисуются прозрачные объекты.

шаг 5. снимается запрет модификации буфера глубины

glDepthMask(GL_TRUE);

шаг 6. Выключается смешивание glDisable(GL_BLEND).

Результат работы алгоритма изображён на рис. 1.

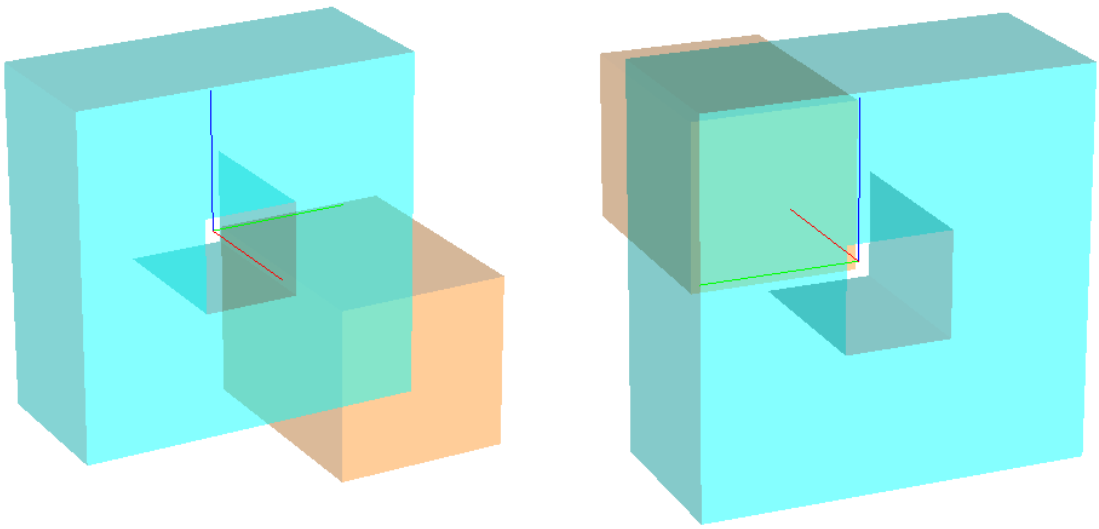

Рис. 1. Результат работы алгоритма прорисовки полупрозрачных элементов.

На рис.1 изображена сцена из двух полупрозрачных кубиков, один из которых имеет отверстие в виде кубика, для двух расположений неподвижного наблюдателя (пользователя ЭВМ) относительно рассматриваемой сцены.

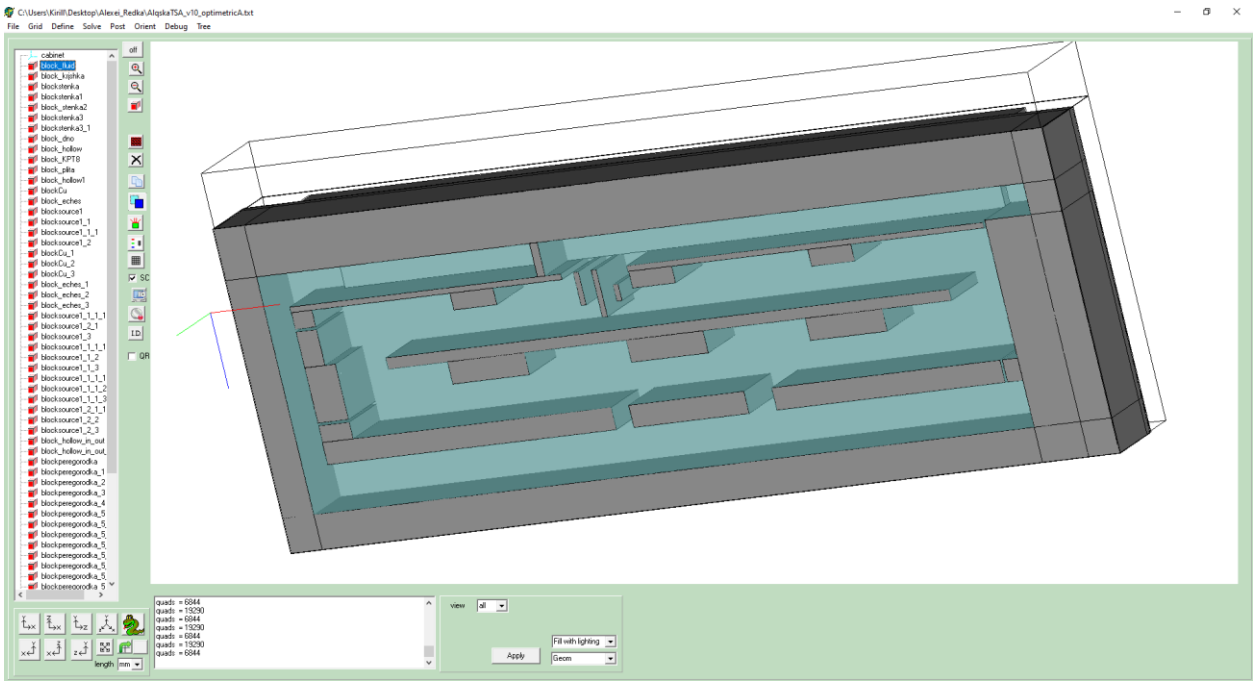

Рис. 2. Результат работы алгоритма прорисовки полупрозрачных элементов.

На рисунке 2 вода в радиаторе водяного охлаждения сделана полупрозрачной.

2. Алгоритм удаления невидимых линий для одиночного кубика с использованием Z - буфера.

Под Z-буфером понимается нахождение точек сцены, ближайших к наблюдателю, и для каждого пикселя изображения отображение только таких точек.

Алгоритм Z буфера был предложен Эдвином Катмуллом (Edwin Catmull) в мае 1975 года. По другим данным ещё раньше этот алгоритм упоминался в диссертации Вольфганга Штрассера в 1974. Его программная реализация, с аппаратным ускорением, стала доступна в библиотеке OpenGL, согласно функциональной спецификации [5] стр. 97, начиная с первой же версии, вышедшей в июле 1994 года. Суть алгоритма с примером доступно изложена в [2] на стр. 321-329. Покажем, как применить данный алгоритм для простейшей сцены прорисовки кубика с использованием OpenGL.

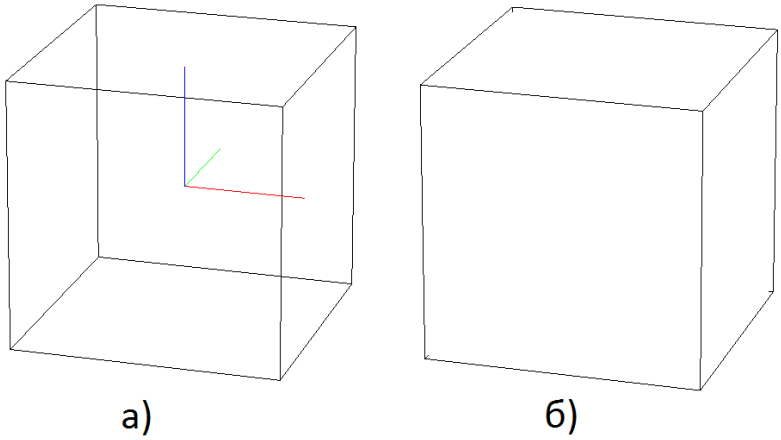

Рис. 3. Визуализация кубика : а) прорисовка всех линий, б) результат работы алгоритма прорисовки каркасной модели с удалением невидимых рёбер.

Алгоритм прорисовки каркасной модели кубика с удалением невидимых рёбер.

шаг 1. Включить Z буфер glEnable(GL_DEPTH_TEST)

(инициализация z-буфера и очистка z-буфера)

и залить всю сцену цветом фона.

шаг 2. В произвольной последовательности граней куба выводить на прорисовку : текущую грань куба, залитую цветом фона (белым), и потом её рёбра черного цвета, но так, чтобы рёбра отстояли от периметра грани во внешнюю сторону на величину 1% от характерного линейного размера текущей грани (качество прорисовки линий зависит от этой константы). И так в цикле по всем граням для каждой грани и её рёбер в произвольной последовательности порядка обхода

граней.

шаг 3. Выключить Z буфер.

Прорисовка граней цветом фона, а также её рёбер черным цветом, отличным от цвета фона, производится для каждого пиксела результирующего изображения с учётом избирательности, на основе значений в z-буфере. Прорисовка граней цветом фона организована для того чтобы удалить те из рёбер куба, которые экранируются текущей гранью.

Результат работы данного алгоритма проиллюстрирован на рис.3.б.

3. Реализация CSG на прямоугольных параллелепипедах.

CSG (*Constructiv Solid Geometry*), или, по-русски, *конструктивная блочная геометрия*, реализована во многих САПР, например, таких как Solid Work. Здесь рассматривается сильно упрощённая реализация, реализованная в [1].

CSG главным образом необходима для визуализации отверстий в сценах, через которые видны другие элементы сцены. Данная упрощённая реализация основана на том, что как сами объекты (solid) так и отверстия (hollow) есть та или иная комбинация прямоугольных параллелепипедов (это верно только для реализованного пакета программ [1]). Таким образом, приходим к алгоритму реализации CSG:

Алгоритм реализации CSG в пакете программ [1].

шаг 1. Завести три одномерных вещественнозначных массива (по одному для каждого координатного направления).

шаг 2. Заносить в каждый из массивов уникальные позиции начала и конца каждого как solid, так и hollow блока для данного координатного направления.

шаг 3. На основе трёх полученных массивов, уникальные значения в каждом из которых предварительно отсортированы по возрастанию пирамидальной сортировкой, сформировать новое solid множество, не содержащее hollow объектов. Свойства нового solid множества таковы, что суммарный объём занимаемый новыми solid-блоками остался неизменным, но зато возросло общее число solid элементов и каждый из новых solid элементов, возможно, составляет только часть размера порождающего его solid элемента.

шаг 4. Прорисовывать только новое solid множество, причём для экономии ресурсов графического ускорителя строго внутренние грани (общие для двух контактирующих по данной грани solid объектов одинакового материала) прорисовке не подвергаются. Также, из оставшихся граней можно прорисовывать лишь те, внешняя нормаль к которым из центра текущего solid элемента составляет острый угол с вектором, начало которого

находится в центре грани, а конец в глазу наблюдателя данной сцены (пользователя ЭВМ смотрящего на монитор). Для того чтобы осуществить последнее, программисту достаточно включить в коде glEnable(GL_CULL_FACE) и все не лицевые грани прорисовываться не будут, т.е. не надо программировать вычисление скалярных произведений самостоятельно для определения того острый угол или тупой.

Также скорость отрисовки зависит от числа прямоугольных граней подлежащих отрисовке. Для сокращения числа прямоугольных граней применяется слияние некоторых прямоугольников нового solid множества, прошедших проверку на одинаковость условий отрисовки, в новые прямоугольники большей площади. При объединении алгоритм ищет кандидата-контейнер с максимальной площадью в которого будут объединяются исходные прямоугольники (оптимизация). Первоначальные мелкие прямоугольники вошедшие в объединение делаются неактивными при отрисовке и рисуется только результирующий прямоугольник большей площади.

С помощью данного алгоритма было реализовано прямоугольное отверстие, изображенное на рис. 1, через которое виден кусок сцены, расположенный за отверстием.

Объекты сцен отличные от прямой прямоугольной призмы, например, цилиндр или плоский полигон, представляют собой набор кубических ячеек, так чтобы криволинейная граница, например, цилиндра, аппроксимировалась ступеньками. В результате получается большее число кубических ячеек, но алгоритм отрисовки не претерпевает изменений. Результаты визуализации сцен с объектами в виде цилиндров или полигонов представлены на рисунках 4 – 5.

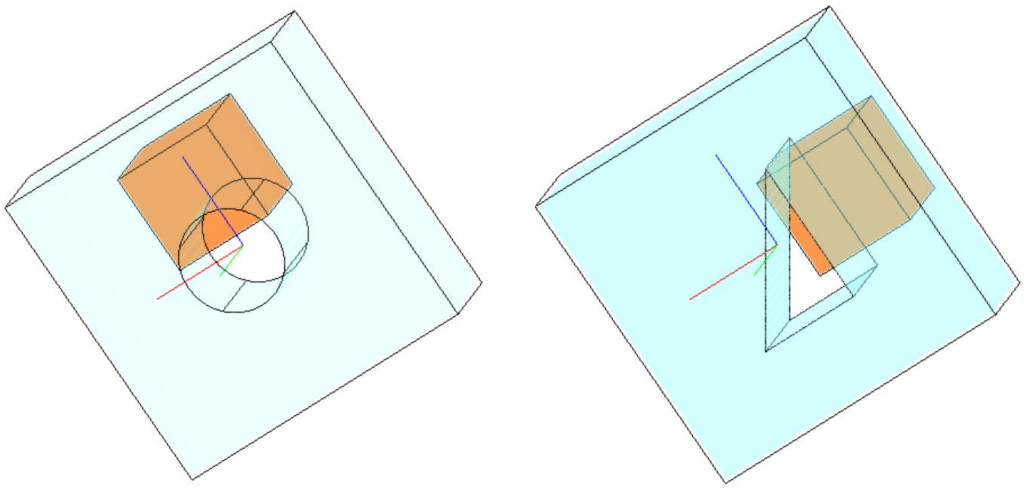

Рис. 4. Отрисовка круглого и треугольного отверстий, реализованным алгоритмом CSG.

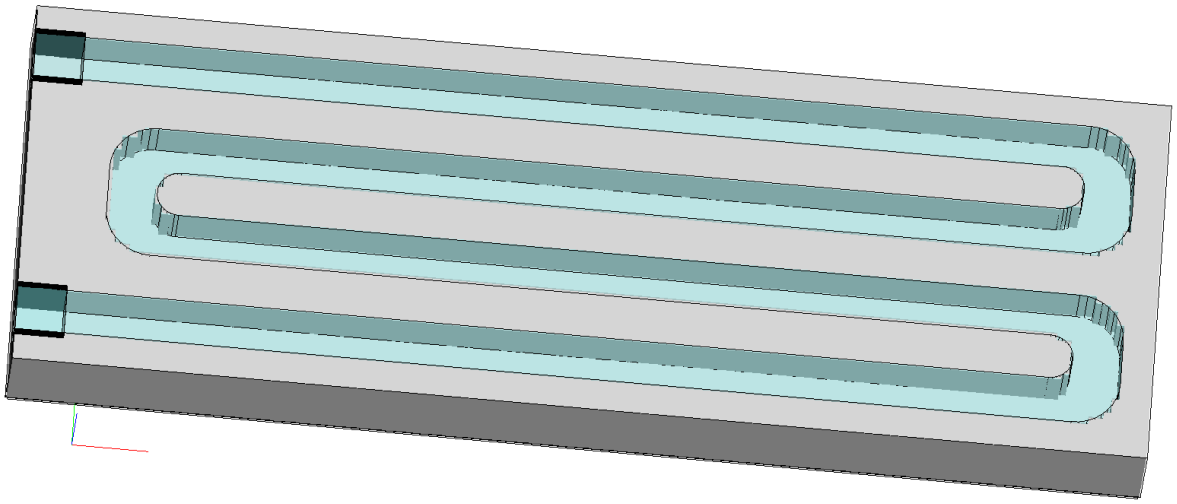

Рис. 5. Отрисовка прозрачного тракта воды в радиаторе водяного охлаждения.

Также можно приписывать цвет не телам, из которых состоит модель для расчёта, а узлам сетки, в которых находилась рассчитываемая величина (температура). Это даёт возможность отобразить на поверхности модели поле температур см. Рис. 6.

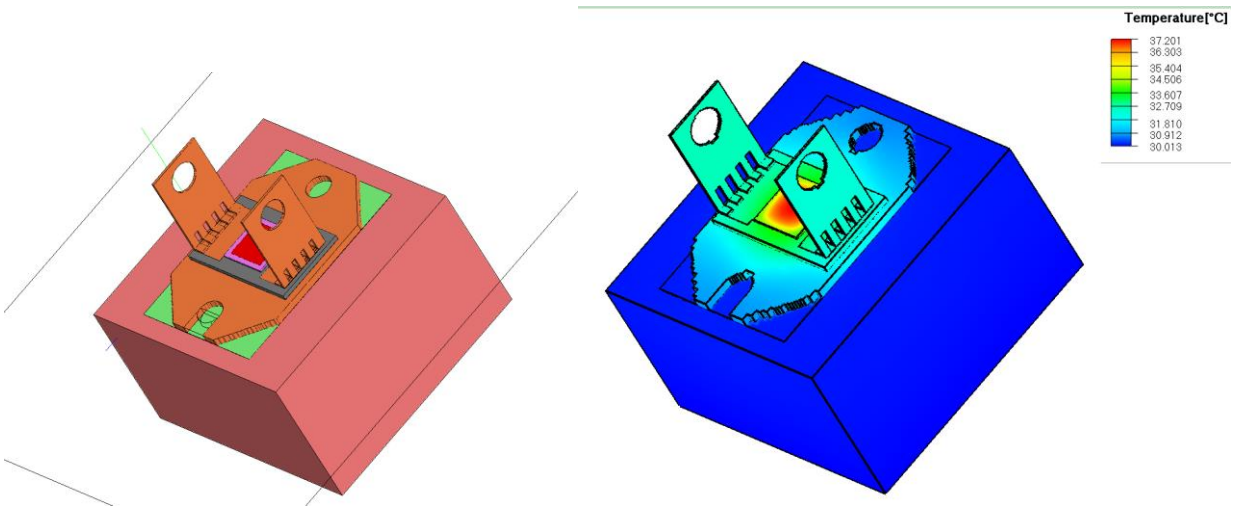

Рис. 6. Диод и поле температур на его поверхности.

4. Модификация алгоритма удаления невидимых линий из п.2 с учётом реализованной CSG.

Так как рёбер в преобразованном solid множестве (из-за CSG) стало значительно больше, то все их прорисовывать ненужно. В [1] применяется следующий избирательный алгоритм:

Модификация алгоритма удаления невидимых линий п.2.

шаг 1. Для преобразованного solid множества из п.3 завести для каждого ребра каждой грани булев индикатор, начальное значение которого задать OFF_VISIBLE (невидимый).

шаг 2. Просканировать рёбра исходного solid и hollow множеств из п.3. Если ребро из шага 1 данного алгоритма является частью текущего ребра исходного solid и hollow множеств при сканировании (организован перебор в двойном цикле), то соответствующий индикатор сделать ON_VISIBLE (видимый).

шаг 3. Запустить алгоритм п.2 применительно к каждому кубу в преобразованном solid множестве из п. 3 с учётом того, что ребра на гранях рисуются чёрным цветом только в случае включенного индикатора, принимающего значение ON_VISIBLE. Грани, залитые цветом фона, рисуются обязательным образом в любом случае (это необходимо для стирания невидимых рёбер закрываемых данной гранью) и конечно до момента прорисовки рёбер на данной грани, если они помечены ON_VISIBLE.

Результат работы данной модификации представлен на рис. 7.

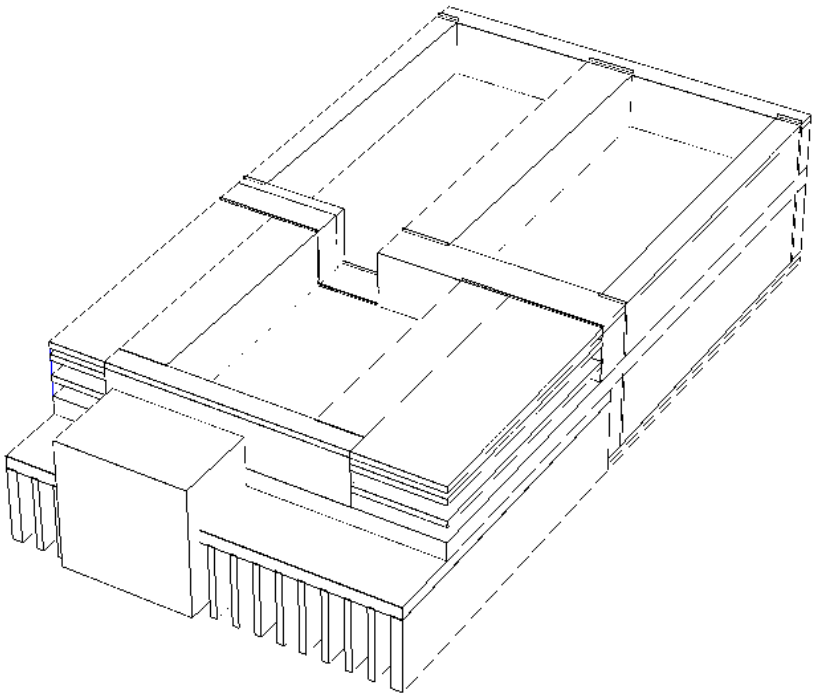

Рис. 7. Изображение радиатора воздушного охлаждения из статьи [4] со снятой крышкой, полученное в программе [1] путём применения алгоритма удаления невидимых линий.

Приведём ещё несколько примеров работы итогового алгоритма отрисовки:

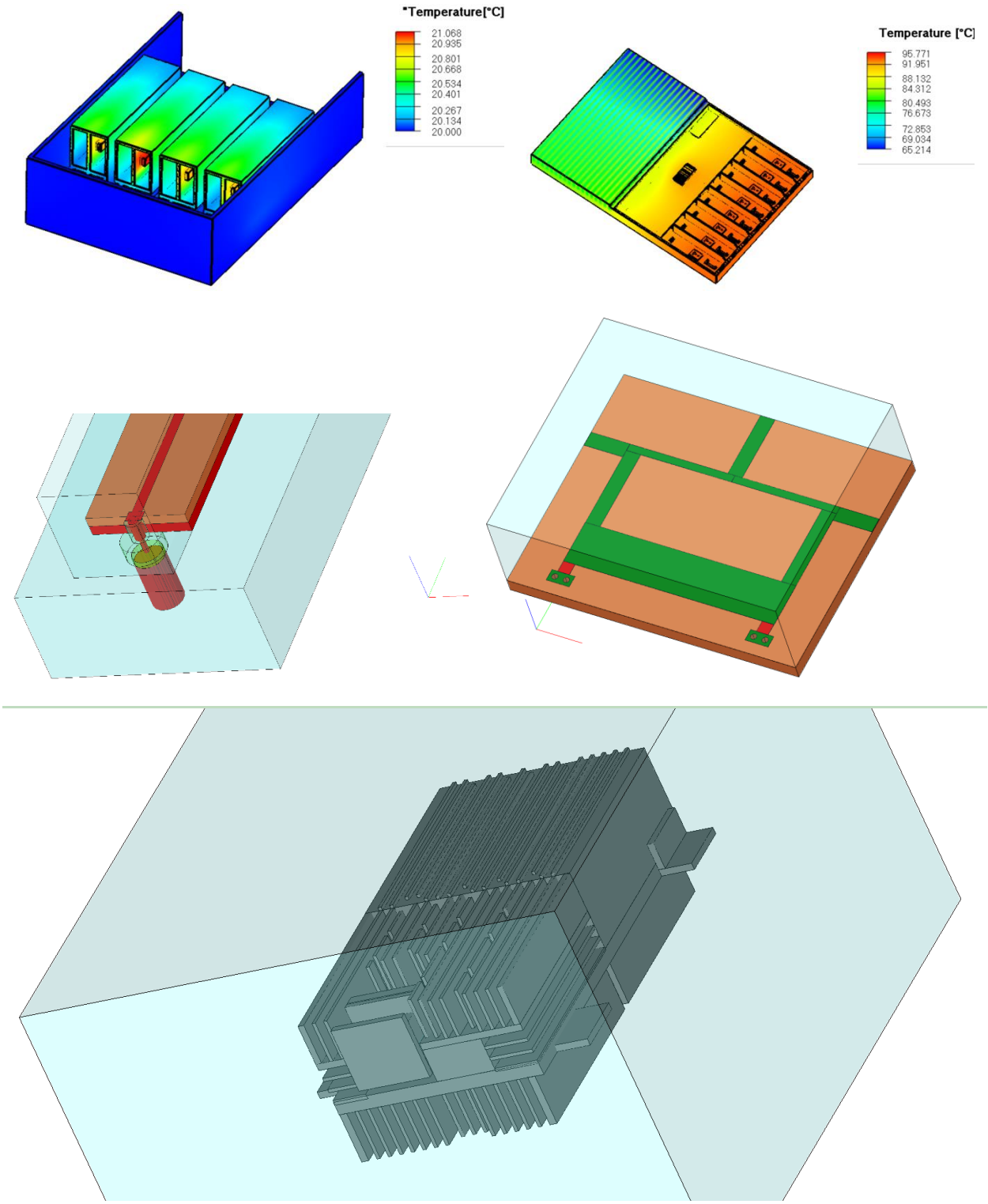

Рис. 8. Демонстрация работы итогового алгоритма отрисовки.

5. Алгоритм вращения модели мышью.

Вращательное движение твёрдого тела в трёхмерном пространстве имеет все три степени свободы. Позиционно-указательное устройство мышь имеет лишь две степени свободы. Чтобы реализовать мышью все три степени свободы вращательного движения

нужна специальная вспомогательная функция GetOGLPos которая преобразует оконные координаты (позицию курсора мыши) в сценовые (трёхмерные координаты) соответствующие той точке в которой находился курсор позиционирующего устройства. Это возможно при использовании алгоритма Z-буффера.

Теперь методика вращения не вызывает вопросов. Цент вращения есть центр кабинета. Другие две точки, первая получена по нажатию левой кнопки мыши с применением GetOGLPos(), вторая аналогичным образом обрабатывает текущую позицию курсора при нажатой левой кнопке. Как только левая кнопка отпущена вращение прекращается. Три точки позволяют получить два вектора с общим началом в центре кабинета. Вращение происходит вокруг оси - векторного произведения этих двух векторов по правилу буравчика. Смена знака векторного произведения меняет ориентацию оси вращения на противоположную и вращение происходит в обратном направлении естественным образом. Компоненты вектора –оси вращения, разлагают угол вращения покомпонентно, на три угла вращения вокруг осей декартовой прямоугольной системы координат. Не стоит забывать постоянно нормировать все длины векторов на единичную длину вектора по евклидовой норме.

Основная содержательная нагрузка хранится в преобразовании оконных координат в сценовые с помощью GetOGLPos().

Выводы

Приведён алгоритм прорисовки полупрозрачных объектов средствами библиотеки OpenGL. Приведён алгоритм реализации отрисовки каркасной модели с удалением невидимых линий. Приведены детали простейшей реализации CSG. Данная статья может быть интересна той категории программистов, которые разрабатывают собственные САПР или студентам технических вузов проходящих курс математических основ машинной графики.

Алгоритмы, описанные в данной статье, реализованы в пакете программ [1], который используется в СВЧ лаборатории Зубкова А.М. для проведения тепловых расчётов полупроводниковых приборов.

С помощью описываемого программного обеспечения [1] могут быть визуализированы компьютерные модели устройств, аналогичные изображенным на рисунке 8, состоящие из булевой комбинации прямых прямоугольных призм, цилиндров или плоских полигонов, начерченные в графическом редакторе программы [1]. Также с помощью описываемой программы [1] могут быть отображены трёхмерные физические поля, например, тепловые, хранящиеся в ASCII формате .PLT. Программой [1] поддерживается считывание трёхмерных рассчитанных физических величин из .PLT файла и их визуализация средствами библиотеки OpenGL.

Обсуждение

Если посмотреть на рис.7, то можно увидеть, что многие линии прорисованы недостаточно чётко. Более чёткая прорисовка получится если более тщательно подобрать константу, фигурирующую в алгоритме п.2.

После реализации в коде замечания к шагу 4 алгоритма п.3 скорость прорисовки достаточно больших моделей, например, таких как изображённая на рис. 7, стала приемлемой - изображение стало возможно вращать мышью, картинка перерисовывалась (рендеринг) достаточно быстро. Вывод состоит в том, что скорость рендеринга удалось увеличить достаточно значительным сокращением количества отображаемых полигонов. Это было достигнуто формированием списка только тех граней которые надо прорисовать. Формирование такого списка занимает существенную долю времени центрального процессора и было вынесено в стадию препроцессинга. Препроцессинг вызывался лишь при изменении пользователем геометрии - удаление или добавление блока, но не при вращении мышью компьютерной модели.

Литература

[1] Иванов К.А., Зубков А.М., Свидетельство о государственной регистрации программ для ЭВМ № 2013660267 "Программа трёхмерного моделирования тепло- и массообмена в радиаторах, состоящих из набора прямоугольных параллелепипедов.

[2] Д. Роджерс, Алгоритмические основы машинной графики. М. МИР, 1989.

[3] М. Краснов, OpenGL графика в проектах DELPHI С-П., "БХВ" 2000.

[4] О.В.Борисов и др. Широкополосный 70-ваттный GaN усилитель мощности X-диапазона. Электронная техника. Серия 2. 2014.

[5] M.Segal, K.Akeley, The OpenGL™ Graphics System : A Specification (Version 1.0), July 1994.