М.Ю. Рудюк, А.М. Кашичкина, А.А. Обухова, А.Р. Салдаева

МЕТОДИЧЕСКИЕ СЛОЖНОСТИ, ВОЗНИКАЮЩИЕ В ПОЛИГОНАЛЬНОЙ ГРАФИКЕ

Россия, г. Пенза, Пензенский государственный технологический университет

The construction of three-dimensional scenes in polygonal graphics faces a number of problems and the authors review them. We are talking about the loss of accuracy when copying a previously prepared raster to the screen, computational instability when using Euler angles, inflexibility of tree structures, limited manufacturability of using the current transparency model and methods of shading.

Полигональная графика является гибким и мощным средством описания трехмерных сцен, позволяющим отобразить практически любой эффект, тем не менее, её использование сопряжено с рядом сложностей, часть которых становятся менее существенными по мере роста вычислительных мощностей, а часть остаются весьма актуальными.

Первой проблемой вычислительной техники, которую хотелось бы упомянуть, являются погрешности и ошибки, возникающие при ***рендеринге*** (отображении) ранее созданного и сохраненного образа на экран.

Очень часто в компьютерной графике используют заранее подготовленные изображения, сохраненные в виде массива точек – растровых изображений. Известно, что это часто позволяет сэкономить вычислительные мощности, когда построения на экране происходят в режиме реального времени. Данное обстоятельство особенно существенно было во времена ограниченных мощностей компьютеров.

Если используемое растровое изображение скопировать на экран без изменений, в режиме 2D-графики, то единственным дополнительным аспектом, который необходимо учитывать, будет возможная **обрезка** тех частей изображения, которые выходят за пределы экрана.

В том случае, если используется 3D-графика, потенциально возникает множество артефактов, ухудшающих результат копирования.

Проблемы возникают, и если разрешение экрана меньше, чем разрешение исходного растра, и в обратном случае. Если исходный растр более подробный, чем это необходимо при отрисовке, то возникает **мерцание**, когда в каждом последующем кадре цвет многих пикселей будет резко меняться. Это возникает из-за того, что при выборке из-за погрешностей округления будут использоваться те или иные пиксели, меняющиеся каждый кадр.

Если разрешение экрана больше, чем используемый в качестве источника данных о цвете растр, то возникает **блочность**, природа которой хорошо понятна, поскольку это, по сути, просто масштабированный без сглаживания исходный растр.

Естественно, для решения данных проблем используются хорошо отработанные техники. Во-первых, необходимо сказать о **MIP – текстурировании** [4], когда автоматически создаются и используются растры нужного разрешения, отличающиеся по размеру в 2^n-раз. Так можно в значительной степени сгладить проблему «мерцания пикселей». Во-вторых, давно применяется техника **фильтрации текстур**, когда при выборке из первоначального растра учитывается цвет не только данного пикселя, но и его соседей. Т.е. цвета нескольких пикселей усредняются и это, во многом, позволяет избежать блочности. Есть несколько способов фильтрации, различающихся количеством используемых соседних пикселей и функцией, применяемой для

усреднения: билинейная, трилинейная, анизотропная. Здесь важно учитывать, что фильтрация текстур не улучшает качество отрисовки, она лишь помогает бороться с ее погрешностями в определенных условиях.

Интересно, что если мы попытаемся смоделировать плавное движение объекта, то ситуация будет различной в случае точного копирования растра (2D-графика) и копирования с пересчетом (3D-графика). В первом случае движение возможно только прерывистое, но результат копирования будет точным. Во втором случае возможно плавное движение, но результат отрисовки никогда не будет полностью идентичным исходному растру. Впрочем, оба эффекта нивелируются по мере увеличения разрешения экранов мониторов.

Вторая проблема вычислительного характера, которая существенная, в частности, при 3D-моделировании, это ***поворот в трехмерном пространстве***. При повороте на плоскости мы пользуемся понятиями центра вращения и угла поворота и этого достаточно для решения практических задач. Если от плоскости перейти к пространству, то здесь необходимо учитывать уже три угла (углы Эйлера [5]). При этом возникает неоднозначность, связанная с последовательностью поочередного вращения на каждый из трех углов, т.е. **некоммутативность**. Следствие этого, как правило, невозможность корректного вращения тела вначале на один угол (ненулевой по всем трем составляющим α_1, β_1, γ_1), а затем на второй угол (α_2, β_2, γ_2). Т.е. полученный результат будет отличен от единичного вращения на угол ($\alpha_1+\alpha_2$, $\beta_1+\beta_2$, $\gamma_1+\gamma_2$). Еще одна сложность – практическая невозможность корректной интерполяции между двумя углами поворота, если они заданы углами Эйлера. Можно провести небольшой эксперимент в одном из трехмерных редакторов, например, 3DS Max: попробовать вращать трехмерную модель. Необходимо выставить углы вращения по осям X и Y по 90°, а затем начать вращать по третьей оси Z. При этом мы обнаружим, что даже при небольших углах вращения (Z) и внешней стабильности объекта видимые в диалоговом окне углы вращения по осям X и Y начнут резко колебаться, принимая значения 90°, -90°, 180°, -180°. Это позволяет говорить о **вычислительной неустойчивости** при использовании углов Эйлера. Хотя на самом деле причина резкого колебания показаний в диалоговом окне немного другая. Очевидно, разработчики 3DS Max заложили в механизм этой программы использование **квартернионов** [3]. Вращение с помощью квартерниона заключается в использовании только одного угла, а также оси вращения (вектор, задаваемый с помощью трех чисел). Применение квартернионов решает все вышеописанные проблемы, возникающие при использовании углов Эйлера (комбинирование углов, интерполяция). Единственная сложность заключается в том, что для нас удобнее и естественнее использовать именно углы. Как правило, в своей деятельности человек использует ортогональные плоскости, как правило, горизонтальную, и один угол поворота. Этого достаточно для решения подавляющего большинства задач, например, ориентации на местности. При необходимости можно добавить еще два угла, например, для характеристики углового положения вертолета.

Далее хотелось бы остановиться на некоторых аспектах применения такой методики обработки информации, как применение **древовидных структур** [2]. Хорошо известен пример использования этой технологии при отрисовке трехмерных сцен в играх Doom и Quake, вышедших в начале-середине 1990 гг. В данном случае древовидные структуры использовались для уменьшения количества отрисовываемых деталей по довольно простому принципу: «если комната невидима, то и объекты, находящиеся в ней тоже не надо отрисовывать». Т.е. если правило верно для узла более высокого уровня, то оно будет верно и для присоединенных к нему дочерних узлов меньшего порядка. Применение древовидных структур позволяет с относительно небольшими вычислительными затратами просчитывать большие объемы данных. Есть несколько

вариантов древовидных структур, которые немного отличаются способом разбиения пространства. Определенная проблема здесь заключается в учете динамических объектов, которые в разные моменты времени могут принадлежать разным узлам. Далее, при регулярном разбиении пространства, можно столкнуться с тем, что один объект может формально относиться к разным узлам.

Следующей до конца нерешенной проблемой компьютерной графики является отрисовка **полупрозрачных объектов** в трехмерных сценах. При отрисовке 2D – изображений серьезных проблем не возникает, необходимо лишь отрисовать слои в правильном порядке. Сложности возникают при использовании буфера глубины и достаточно детализированных моделей. Базовый алгоритм, как известно, заключается в следующем: нужно отрисовать вначале непрозрачную геометрию, а затем прозрачные элементы в порядке от дальнего к ближнему. При этом полупрозрачный пиксель будет отрисовываться, если он находится в пространстве сцены ближе, чем данные в буфере глубины. При этом используется хорошо известная формула суммирования полупрозрачного объекта с фоном, исходя из текущего значения **альфа-канала** [1]. Если два полупрозрачных объекта пересекаться, то при сортировке по дальности на уровне объектов корректное изображение получить будет невозможно. Конечно, возможна сортировка на уровне полигонов, но это связано с дополнительными неудобствами, хотя бы потому, что полигонов может быть много и эта сортировка будет связана с дополнительными вычислительными затратами. Далее, два полигона могут пересекаться. Эта ситуация вовсе не имеет решения, если, конечно, не рассматривать возможность сортировки отдельных пикселей, что, очевидно, будет связано с неприемлемыми временными затратами. В реальности данные артефакты, связанные с сортировкой, возникают редко, однако, сам факт сортировки и необходимость рендеринга в несколько проходов вызывает некоторое неудовольствие у программистов. Обойти необходимость этих дополнительных усилий при используемой модели полупрозрачности практически невозможно. Необходимо упомянуть, что предлагались решения данной проблемы, т.е. алгоритмы Order-Independent Transparency, которые формально позволяют отрисовывать всю геометрию за один проход, но, как правило, они достаточно сложны, громоздки, потенциально могут создавать артефакты и ограничивают возможности программиста реализовывать другие сложные графические эффекты на современном графическом API. Принципиальным решением было бы применение рейтрейсинга, который многие графические эффекты воспроизводит очень близко к реальности.

Схожую по природе с предыдущей проблемой сущность имеют сложности при отрисовке ***теней*** в трехмерных сценах. Если рассматривать тени в объективном физическом мире, то там это просто участки пространства, куда не проникает свет. Отрисовка трехмерной сцены осуществляется со многими условностями, физические эффекты эмулируются с той или иной точностью. Основа полигональной графики – растеризация треугольника и при этом расчет освещенности сам по себе не предполагает такого явления, как тени. Тем не менее в компьютерной графике довольно давно научились формировать тени от объектов, в том числе динамических. Первый вид теней – **стенсильные**. Они строятся с использованием специального т.н. буфера трафарета, который позволяет быстро осуществлять многие высокозатратные вычислительные операции. При этом строится вспомогательная сетка как проекция данного объекта на ту поверхность, где должна быть тень, из положения источника света. Стенсильные тени используются все меньше, поскольку, в частности, не позволяют учитывать полупрозрачность и с помощью них трудно получать «мягкие», размытые тени. Интересно, что объект, который должен отбрасывать стенсильную тень, должен быть замкнутым, т.е. образовывать закрытую трехмерную поверхность.

Следующий вид теней получают с помощью «**теневых карт**». Вначале делается отрисовка сцены с позиции источника света, а далее вторая отрисовка уже с позиции наблюдателя и при этом накладывается ранее полученная теневая карта. Данный способ построения теней более универсальный, хотя теневые карты часто получаются с артефактами. И стенсильные тени, и теневые карты требуют дополнительных манипуляций. Более просто получать тени с помощью рейтрейсинга. При этом реализуется физически реалистический алгоритм получения теней, когда затененный пиксель получается, когда луч встречает препятствие.

1. Альфа-канал // Wikipedia [сайт], URL: https://ru.wikipedia.org/wiki/Альфа-канал
2. Древовидная структура // Wikipedia [сайт], URL: https://ru.wikipedia.org/wiki/Древовидная_структура
3. Кватернионы и вращение пространства // Wikipedia [сайт], URL: https://ru.wikipedia.org/wiki/Кватернионы_и_вращение_пространства
4. Теневая карта // Wikipedia [сайт], URL: https://ru.wikipedia.org/wiki/Теневая_карта