Вопрос № 1

Понятие алгоритма. Свойства. Способы записи

1. Алгоритмом называется точное предписание, определяющее точное содержание и порядок действий, которые необходимо выполнить над исходными данными для получения конечного результата при решении всех задач определенного класса.

Свойства алгоритма:

1. Массовость (для решения целого количества задач)
2. Понятность
3. Дискретность
4. Конечность
5. Определенность (при исполнении алгоритма исполнитель должен соответствовать с заданными действиями)
6. Эффективность.

Способы записи алгоритма.

1) Словесная запись.

Это перечисление словесных действий, которые необходимо выполнить в определенном порядке.

Пример:

$$y = \begin{cases} -2, & x < 1 \\ -x + 3, & x \geq 1 \end{cases}$$

1. Ввести значение x
2. Если x<1, то y=-2, иначе y=-x+3
3. Печать y
4. Конец

2) Решающая таблица

Состоит из 4-х частей: перечень условий, перечень действий, указатель условий, указатель действий.

Таблица	1	2	...	K	
Условие 1	Y	N	...	Y	указатели условий
...					
Условие N	N	Y	...	N	
Действие 1	X	...	...	...	указатели действий
...					
Действие 2	...	...	...	X	

Содержание столбца составляет правила алгоритма, определяющие какие условия следует проверить, каким должен быть результат проверки и какие действия.

Если условие входит как элемент из правила - ставится Y(yes - условие должно быть выполнено) или N(no - не должно).

В том случае, когда правила требуют выполнения некоторых действий на пересечении строки и столбца ставится X.

Пример:

Таблица	1	2	
x<1	Y	N	Если выполняется условие x<1, то
y=-2	X		y=-2 и печатаем y (столбец 1).
y=x+3		X	Если не выполняется условие x<1, то
печать y	X	X	y=x+3 и печатаем y (столбец 2).

Вопрос № 2

Построение реалистических изображений. Алгоритм построения теней в машинной графике

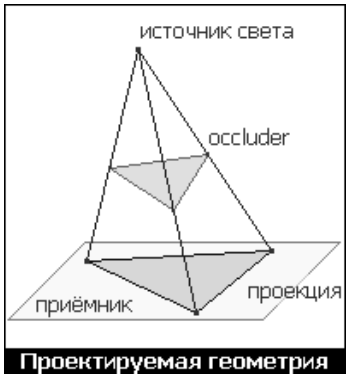
В сфере компьютерной трехмерной графики главным направлением деятельности на данный момент можно назвать задачу увеличения реалистичности изображения. Одной из составляющих этой задачи является визуализация теней. Во многом благодаря теням человеческий мозг получает информацию о взаимном расположении объектов в пространстве. Поэтому отображение теней является тем фактором, который может существенно улучшить реалистичность трехмерных сцен.

Существует довольно много методов рендеринга теней в реальном времени, каждый имеет свои преимущества и недостатки. Выбор алгоритма зависит от нужд и специфики графического приложения. Основными наиболее часто применяемыми методами построения теней являются следующие:

1. фейковые тени
2. метод проекции вершин
3. теневые объемы
4. карты затемнения
5. теневые буферы

1. Фейковая тень – всего лишь полигон с изображением полупрозрачного кружочка, который "кладется" на землю прямо под моделью.

2. Метод проекции вершин –метод также очень прост, но позволяет добиться куда более реалистичных результатов, нежели предыдущий. Суть метода состоит в том в том, что проецируем каждый полигон модели на плоскую землю, а точнее горизонтальную плоскость. Вполне очевидно, что такая тень не будет падать на посторонние объекты сцены и будет верна только для абсолютно ровного пола.



3. Теневые объемы (stencil-тени). Алгоритм был впервые предложен Франклином Кроу в 1977 году для генерации теней в трехмерном пространстве. Особенностями алгоритма можно назвать построение геометрически правильных теней с четкими краями, а также использование буфера трафаретов (stencil-буфера) в современных реализациях (в связи с этим тени построенные данным методом называют stencil-тенями). К достоинствам алгоритма можно отнести также то, что тени можно строить только от необходимых объектов, а не вообще от всех объектов сцены (хотя такая возможность есть).

Буфер трафаретов – дополнительная плоскость в буфере кадра. Stencil-буфер позволяет задать для каждого пикселя экрана какое-нибудь числовое значение (обычно от 0 до 255), которое затем можно использовать, чтобы идентифицировать точки, над которыми необходимо совершить ту или иную операцию. С помощью stencil-буфера можно определить пиксели, которые попали в область геометрической тени, отбрасываемой 3D-объектом и сделать их соответственно темнее.

Алгоритм основан на использовании так называемых "теневых объемов" (shadow volumes). Теневой объем представляет собой область трехмерного пространства, полностью заполненную тенью. Теоретически, теневые объемы строятся по точкам силуэта объекта, отбрасывающего тень, и лучам, исходящим из этих точек по направлению от источника света. Однако практически, вместо лучей используются отрезки конечной длины, так как, во-первых, технически сложно работать с бесконечными величинами, а во-вторых, сцена всегда ограничена пирамидой видимости.

Для нахождения силуэта вычисляем ориентацию всех граней объекта относительно источника света (ориентацию определяет знак скалярного произведения нормали

треугольника и направления на источник света), ищем все ребра, которые лежат между треугольниками разной ориентации. Полученные ребра составляют силуэт. Силуэт вытягиваем в бесконечность (точнее на некоторое заранее заданное большое число). Далее находим часть сцены, расположенную внутри теневого объема. Для этого используем stencil-буфер. Рисуем всю сцену без теней. Теперь два раза рисуем теневой объем. Первый раз рисуем те грани, которые повернуты лицевой стороной к камере, и при этом увеличиваем стенсил там, где пройден Z-test. В результате в стенсиле мы получим значения больше нуля там, где передняя часть объема находится к нам ближе чем к сцене. За второй проход рисуем грани, которые повернуты лицевой стороной от камеры, и уменьшаем значения стенсила там, где пройден Z-test. В результате после обоих проходов мы получим ненулевые значения там где передняя часть объема прошла тест, а задняя - не прошла. На самом деле, это и есть та часть сцены на экране, которая находится внутри объема, а значит - в тени от того объекта, от которого строили теневой объем. Рисуем серый полупрозрачный прямоугольник на весь экран, разрешая закраску только тех пикселей, для которых значения stencil-буфера равны нулю.

4. Карты затенения. Чтобы создать карту затенения (shadow map), необходимо переместить камеру в точку, в которой расположен источник света, и ориентировать ее так, чтобы был виден объект, тень которого мы хотим построить. Очевидно, что из такого положения тень отбрасываемая объектом не будет видна вообще, т.к. она загорожена самим объектом. Затем мы рисуем наш объект, так как он виден из установленной камеры, но только черным цветом на белом фоне, получив, таким образом, картинку (shadow map), которая является как бы поперечным срезом конуса тени всего объекта. Последнее, что остается сделать, так это наложить полученную карту затенения на другие объекты 3D-сцены. Делается это обычным проецированием полигонов.

5. Теневые буферы. Алгоритм впервые был представлен Лансом Вильямсом в 1978 году. Потенциально метод теневых буферов – самый универсальный метод построения теней в реальном времени, хотя его реализация до недавнего времени была затруднена отсутствием необходимого оборудования.

Алгоритм базируется на том факте, что область, находящаяся в тени относительно источника света, не видна из точки его положения. Для начала сцена рисуется из

источника света во внеэкранный буфер. Нас интересуют только z-значения точек буфера. Картина глубин, сохраненная в z-буфере (теневом буфере), будет использоваться для теста на принадлежность точек тени. В буфере записаны глубины всех ближайших к источнику света точек. Любая точка, находящаяся дальше соответствующей точки в теневом буфере, будет в тени. Во время основного рендеринга сцены каждый пиксель, видимый из камеры, должен быть спроектирован на виртуальную плоскость теневого буфера. Если расстояние от точки до источника света равно соответствующему z-значению буфера, то пиксель освещен. Если точка дальше, то пиксель в тени.

Вопрос № 3

Записать программу с применением параллельных вычислений

```
//Текст програмы, исполняющейся на устройстве (GPU или CPU). Именно
 эта программа будет выполнять паралельные
//вычисления и будет складывать вектора. Программа написанна на язы
ке, основанном на C99 специально под OpenCL.
string vecSum = @"
__kernel void floatVectorSum(__global float * v1, __global float *
v2)
{
    int i = get_global_id(0);//Уникальный глобальный ID текущего
элемента вектора
    v1[i] = v1[i] + v2[i];//Результат сложения записываем в вектор
v1
}";

//Инициализация платформы
OpenCLTemplate.CLCalc.InitCL();
//Компиляция программы vecSum
CLCalc.Program.Compile(vecSum);
//Присовоение названия скомпилированной программе, её загрузка.
CLCalc.Program.Kernel VectorSum = new CLCalc.Program.Kernel("floatV
ectorSum");

//Исходный и результирующий векторы
float[]
      v1 = new float[n],
      v2 = new float[n],
```

```
      v3 = new float[n];

//Инициализация векторов, которые мы будем складывать
for (int i = 0; i < n; i++)
{
    v1[i] = i;
    v2[i] = i * 2;
}


//Загружаем вектора в память устройства
CLCalc.Program.Variable varV1 = new CLCalc.Program.Variable(v1);
CLCalc.Program.Variable varV2 = new CLCalc.Program.Variable(v2);

//Объявление того, кто из векторов кем является
CLCalc.Program.Variable[] args = new CLCalc.Program.Variable[] { va
rV1, varV2 };

//Сколько потоков будет запущенно
int[] workers = new int[1] { n };

//Исполняем ядро VectorSum с аргументами args и колличеством потоко
в workers
VectorSum.Execute(args, workers);

//Выгружаем из памяти содержимое вектора v1 как результат сложения
в вектор v3
varV1.ReadFromDeviceTo(v3);
```