Мультитекстурирование

Мультитекстурирование.

Современные видеокарты и OpenGL поддерживают мультитекстурирование – то есть наложение на один объект сразу нескольких текстур. При этом процесс наложения нескольких текстур, так же как и одной текстуры, контролируется тремя основными структурами данных:

1. **Текстурный объект** – определяет саму текстуру, которая может быть наложена на полигон. Текстурный объект хранит в себе изображение текстуры (включая все мипмапы), режим фильтрации, режим адресации текселей и некоторые дополнительные параметры (цвет границы). В случае мультитекстурирования необходимо создать и использовать несколько текстурных объектов!

2. **Текстурный модуль –** определяет какая текстура из ранее загруженных и как будет накладываться на полигон. Текстурный модуль хранит идентификатор накладываемого текстурного объекта, признак разрешения или запрета наложения текстур, способ наложения текстуры, матрицу текстурных координат и некоторые другие параметры (цвет текстурного модуля). В случае мультитекстурирования так же необходимо задействовать несколько текстурных модулей.

3. **Текстурные координаты –** задаются для каждой вершины и указывают соответствие между трехмерным полигоном и изображением текстуры, то есть какой участок текстуры соответствует определенному полигону. В случае использования мультитекстурирования необходимо передавать в OpenGL текстурные координаты для каждого задействованного текстурного модуля.

Мультитекстурирование.

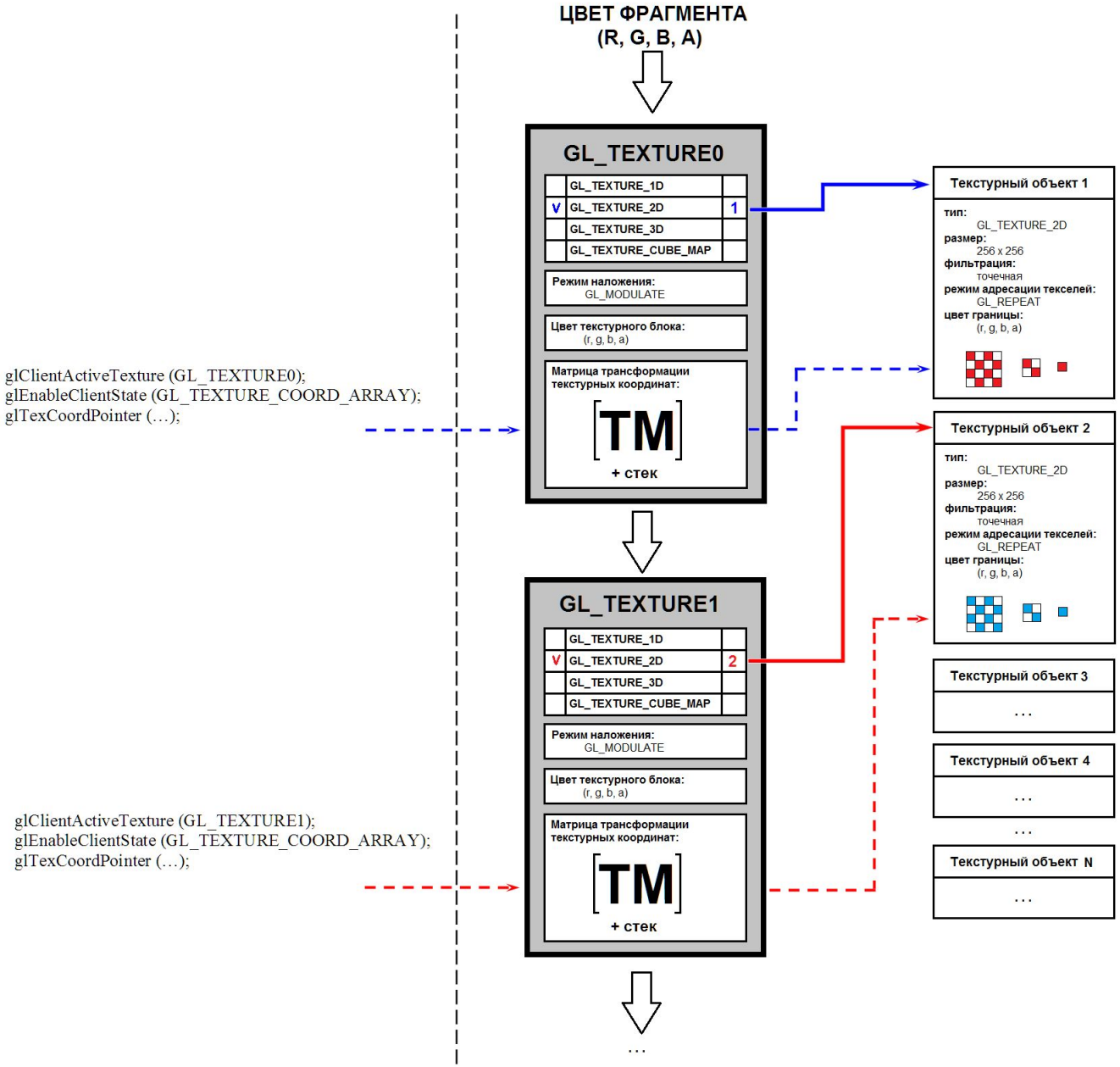
При этом процесс наложения нескольких текстур происходит поэтапно:

1. **Вначале вычисляется цвет фрагмента.** Если режим расчета освещения не используется, то цвет фрагмента определяется по цветам, заданным в вершинах полигона, с помощью линейной интерполяции. Если режим расчета освещения используется, то для каждой вершины вычисляется цвет по формуле Фонга с учетом параметров источника света, материала и взаимного расположения источника света, вершины и наблюдателя. Затем полученные значения интерполируются для каждого фрагмента полигона. Цвет полигона всегда представлен вектором (R, G, B, A).

2. **Затем происходит наложения текстуры в нулевом текстурном блоке.** Если текстурирование в данном текстурном блоке запрещено (по умолчанию), то данный блок пропускается, итоговый цвет на данном этапе не меняется. Если текстурирование разрешено, то:
    1. Извлекаются текстурные координаты соответствующие данному текстурному блоку и умножаются на матрицу текстурных координат данного текстурного блока. Матрица текстурных координат задается индивидуально для каждого текстурного блока и может использоваться для преобразования текстурных координат.
    2. По полученным текстурным координатам идет обращение к текстурному объекту, привязанному к данному текстурному блоку. Текстурный объект с учетом своего режима адресации текселей и режима фильтрации возвращает некоторое усредненное значение из текстуры. Возвращаемое значение всегда представлено вектором (Rt, Gt, Bt, At).
    3. В соответствии с режимом наложения текстуры установленном в этом текстурном блоке вычисляется итоговый цвет как комбинация ранее вычисленного цвета (R, G, B, A) с цветом полученным из текстуры (Rt, Gt, Bt, At). Вычисленный цвет передается следующим текстурным блокам.

3. **Далее происходит наложения текстуры в первом текстурном блоке.** Наложение происходит аналогичным образом, но с использованием параметров данного текстурного блока.

Мультитекстурирование.



Вывод модели. Указание текстурных координаты.

В случае наложения нескольких текстур (то есть использования нескольких текстурных блоков) необходимо передать текстурные координаты для каждого из этих текстурных блоков. Непосредственно задание текстурных координат с использованием массивов вершин и буферов VBO происходит по следующему сценарию:

1. Указываем текущий буфер вершин – буфер в котором хранятся текстурные координаты:

```
glBindBuffer (GL_ARRAY_BUFFER, … );
```

2. Указываем, для какого текстурного блока задаются текстурные координаты:

```
glClientActiveTexture (GL_TEXTURE0);
```

3. Включаем массив текстурных координат. Массив текстурных координат включается именно для ранее указанного текстурного блока. Для всех остальных текстурных блоков состояние включенности/отключености не меняется:

```
glEnableClientState (GL_TEXTURE_COORD_ARRAY);
```

4. Указываем откуда и как извлекать текстурные координаты для ранее выбранного текстурного блока:

```
glTexCoordPointer (…)
```

5. Шаги 2-3-4 необходимо повторить для всех текстурных блоков, которые мы планируем использовать.

Вывод модели. Указание текстурных координаты.

6. Указать откуда следует брать все прочие данные – такие как геометрические координаты, вектора нормалей, цвета и, возможно, еще какие-то атрибуты. Для этого так же необходимо включить соответствующие массивы и использовать соответствующую функцию для указания данных (например - glVertexPointer).

7. Вывести модель, используя, возможно, массив индексов и одну из команд glDrawElements или glDrawArrays.

В результате при выводе модели для нее будут передаваться все ранее включенные массивы, в том числе текстурные координаты для нескольких текстурных блоков!

8. После вывода модели желательно отключить все ранее включенные массивы, поскольку последующие объекты могут не использовать все те же самые атрибуты. В частности, не все модели используют несколько комплектов текстурных координат. Отключение массива текстурных координат производится так же с помощью указания текстурного блока и отключения в выбранном блоке массива текстурных координат:

```
glClientActiveTexture(GL_TEXTURE0);
glDisableClientState(GL_TEXTURE_COORD_ARRAY);
```

Вышеприведенную последовательность необходимо выполнить для всех текстурных блоков, для которых ранее были включены массивы текстурных координат.

Вывод модели. Указание текстурных координаты.

Рассмотрим два способа хранения текстурных координат для наложения двух текстур. В первом случае мы действительно храним два комплекта текстурных координат для каждой вершины:

```
// Структура описывающая одну вершину (каждая вершина имеет геометрические координаты и цвет)
struct ModelPoint
{
      // геометрические координаты
      Glfloat  Coord[3];
      // нормаль
      GLfloat  Normal[3];
      // цвет
      Glubyte  Color[3];
      // текстурные координаты 0
      Glfloat  TexCoord_0[2];
      // текстурные координаты 1
      Glfloat  TexCoord_1[2];
};
```

Тогда их передача в OpenGL при выводе модели выглядит следующим образом:

```
glClientActiveTexture (GL_TEXTURE0);
glEnableClientState (GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer (2, GL_FLOAT,  sizeof(ModelPoint),
(GLvoid*)offsetof(ModelPoint,TexCoord_0));

glClientActiveTexture (GL_TEXTURE1);
glEnableClientState (GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer (2, GL_FLOAT,  sizeof(ModelPoint),
(GLvoid*)offsetof(ModelPoint,TexCoord_1));
```

Вывод модели. Указание текстурных координаты.

Второй способ подразумевает что обе текстуры накладываются либо по одним и тем же текстурным координатам, либо координаты для каждой текстуры каким-либо образом преобразуются с помощью матриц текстурных координат. В этом случае нет смысла хранить второй экземпляр текстурных координат, вместо этого для нулевого и первого текстурного блоков передаются одни и те же значения:

```
// Структура описывающая одну вершину (каждая вершина имеет геометрические координаты и цвет)
struct ModelPoint
{
      // геометрические координаты
      Glfloat  Coord[3];
      // нормаль
      Glfloat  Normal[3];
      // цвет
      Glubyte  Color[3];
      // текстурные координаты 0
      Glfloat  TexCoord[2];
};
```

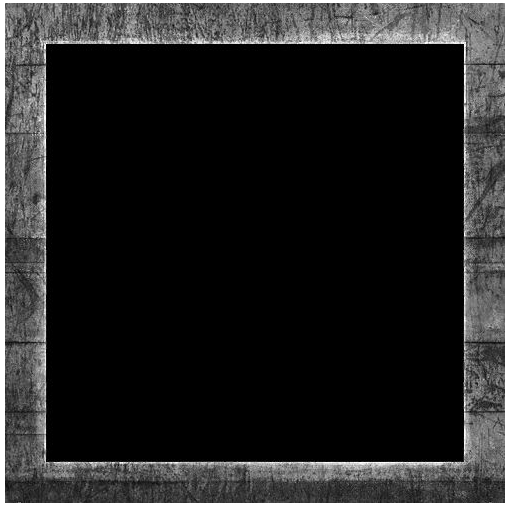
Тогда их передача в OpenGL при выводе модели выглядит следующим образом:

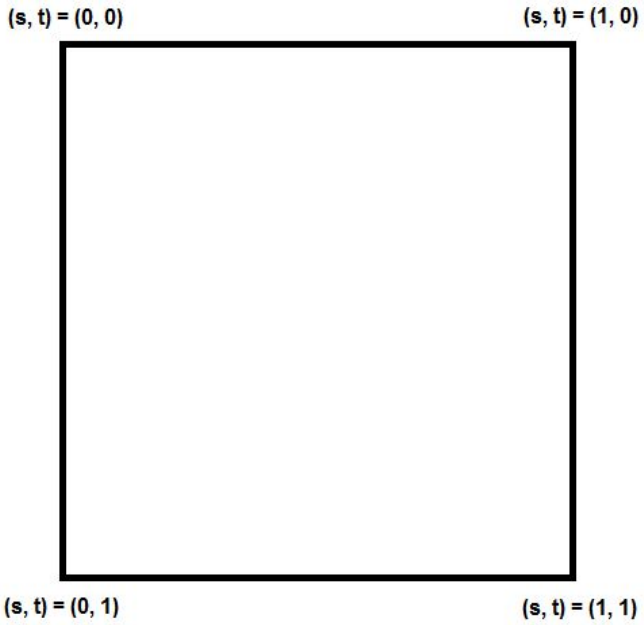
```
glClientActiveTexture (GL_TEXTURE0);
glEnableClientState (GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer (2, GL_FLOAT,  sizeof(ModelPoint), (GLvoid*)offsetof(ModelPoint,TexCoord));

glClientActiveTexture (GL_TEXTURE1);
glEnableClientState (GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer (2, GL_FLOAT,  sizeof(ModelPoint), (GLvoid*)offsetof(ModelPoint,TexCoord));
```

Вывод модели. Указание текстурных координаты.

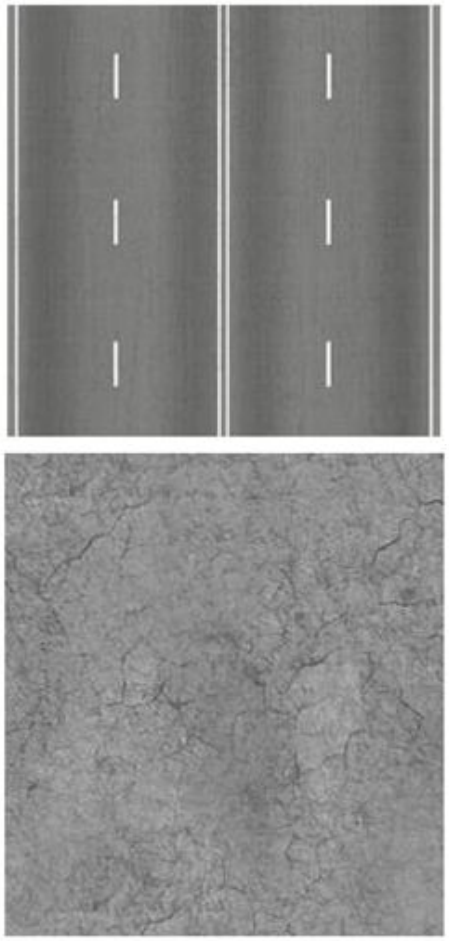
В качестве примера использования одних и тех же текстурных координат можно привести наложение диффузной и зеркальной текстуры:

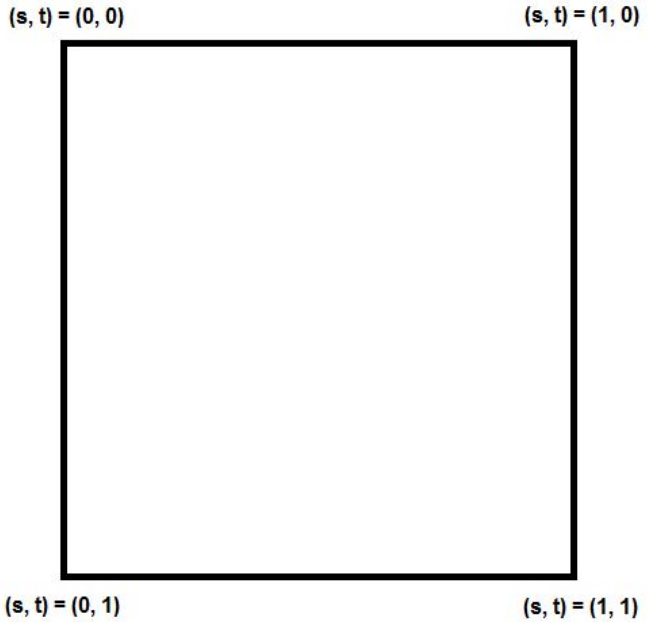




Вывод модели. Указание текстурных координаты.

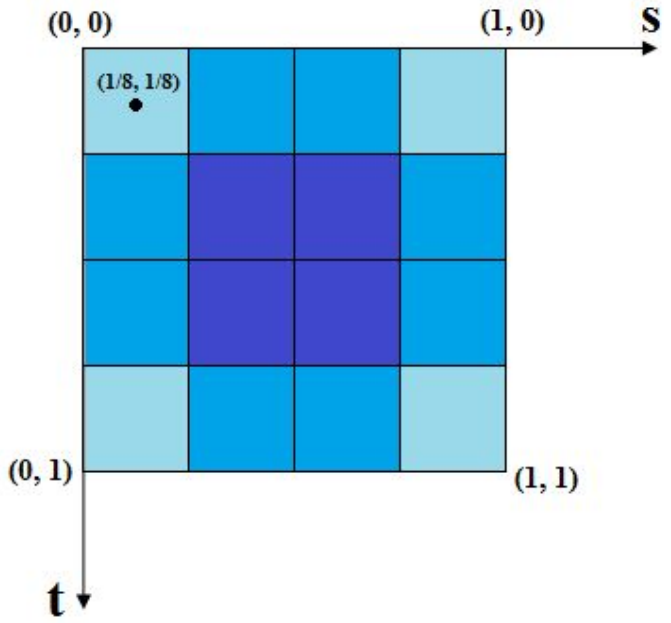
Другим примером является наложение текстуры детализации. В этом случае текстура детализации накладывается не совсем по тем же текстурным координатам, но текстурные координаты для текстуры детализации можно легко получить применив необходимую матрицу текстурных координат:

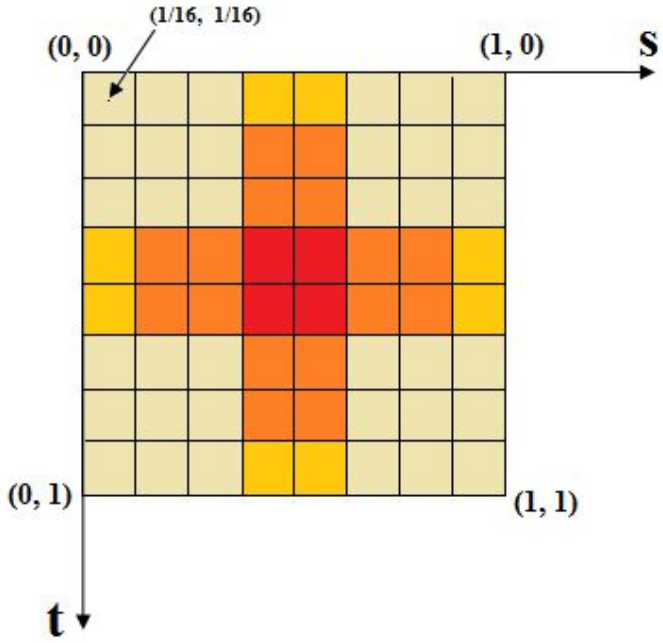




Вывод модели. Указание текстурных координаты.

Для наложения двухмерных текстур всегда используется вектор из двух компонент для представления текстурных координат. В OpenGL такой вектор принято обозначать как (s, t).

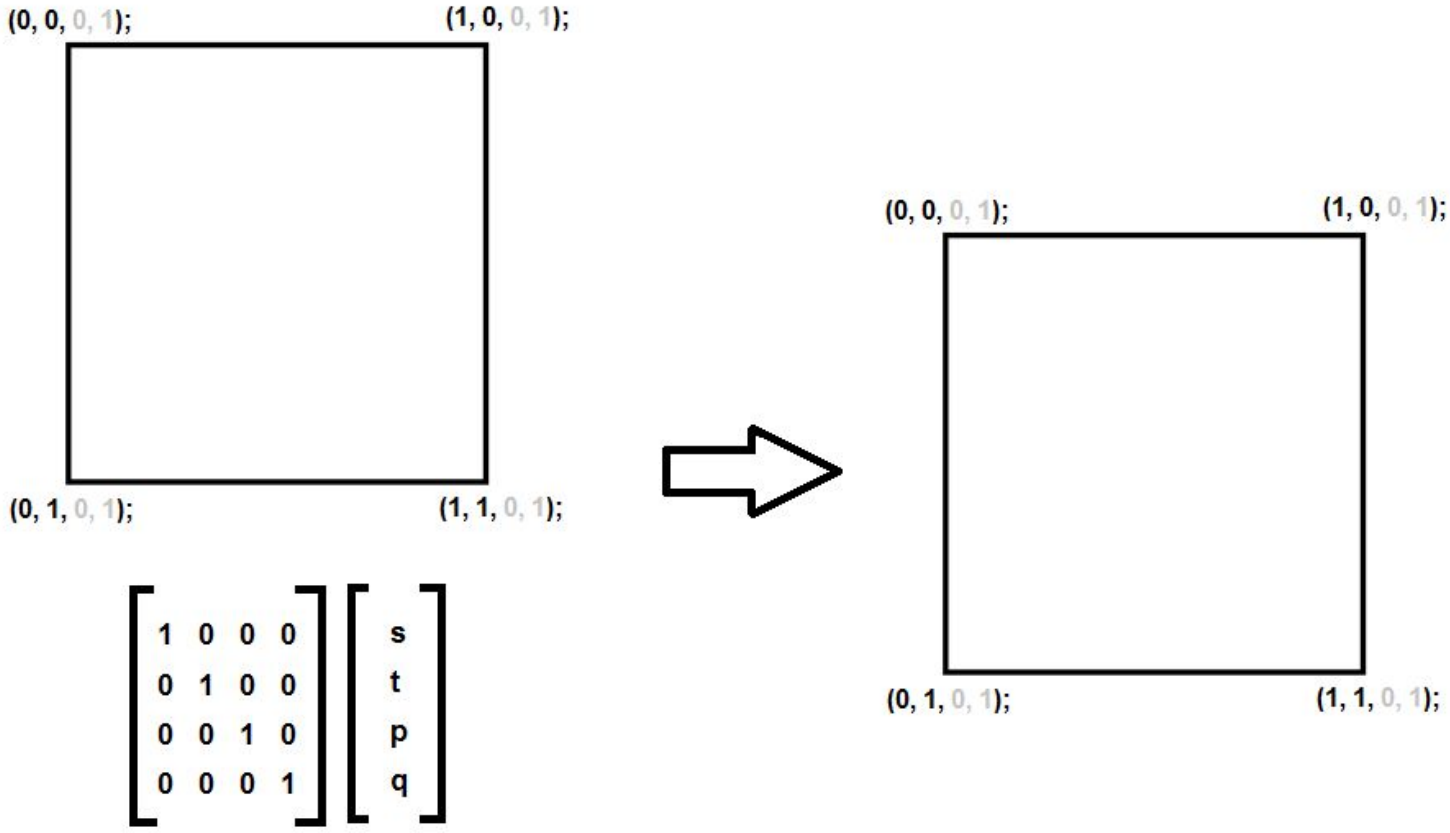




При этом OpenGL позволяет накладывать текстуры и других типов, например трехмерные, поэтому в общем случае для представления текстурных координат используется вектор из четырех компонент обозначаемых как (s, t, r, q) (иногда как – (s, t, p ,q)). В случае, если мы указываем не все четыре компоненты, то оставшимся компонентам автоматически присваивается значение по умолчанию, равное (0, 0, 0, 1).

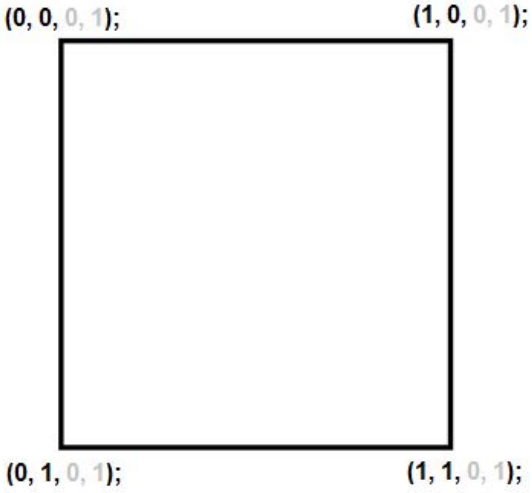
Использование матриц текстурных координат.

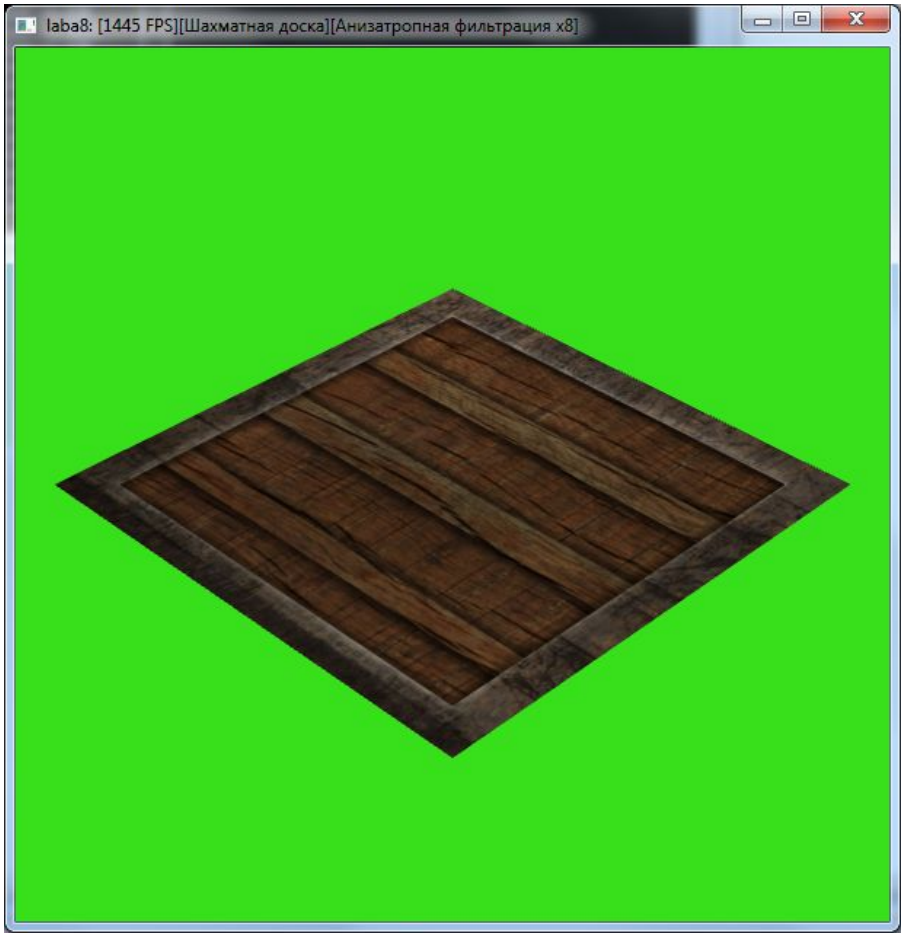
Перед обращение к текстурному объекту для получения значения из текстуры, OpenGL автоматически умножает переданные текстурные координаты на матрицу текстурных координат, заданных для каждого блока. Значение по умолчанию для такой матрицы – это единичная матрица, поэтому текстурные координаты по умолчанию оказываются неизменными:



Использование матриц текстурных координат.

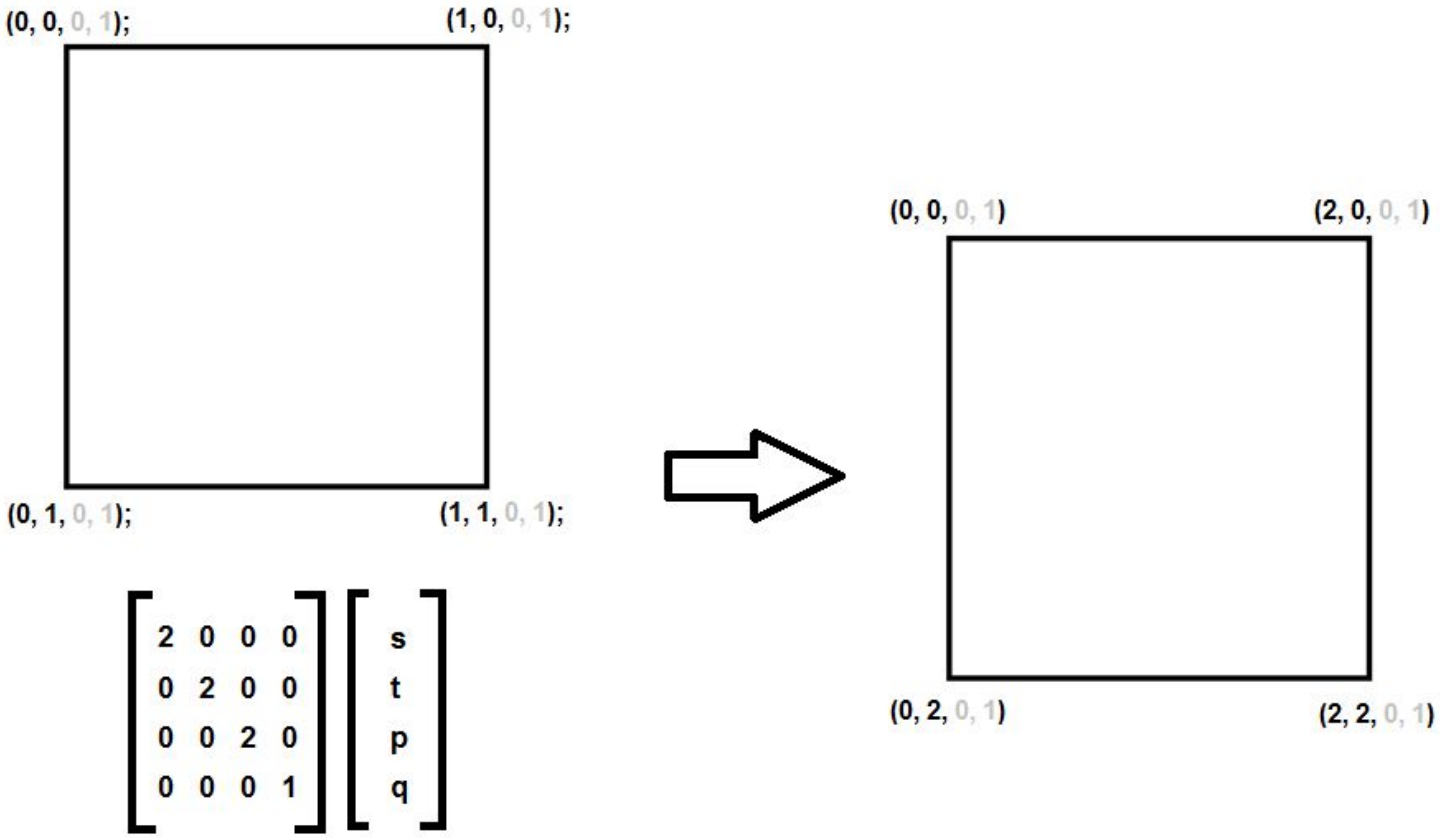
В результате текстура будет наложена по тем же самым текстурным координатам без искажений:





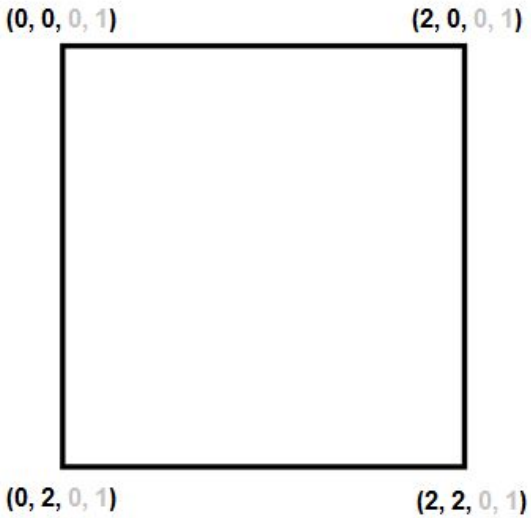
Использование матриц текстурных координат.

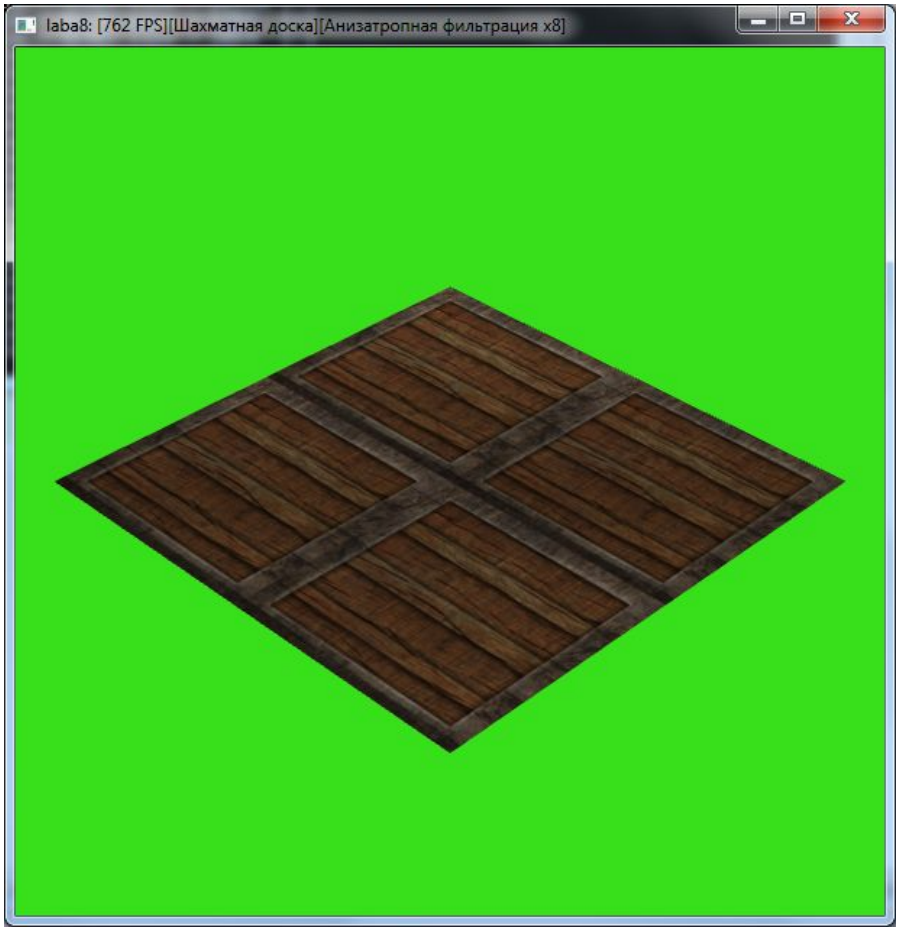
Теперь рассмотрим что будет, если в качестве текстурной матрицы на момент вывода модели была задана матрица приведенная ниже:



Использование матриц текстурных координат.

В результате текстурные координаты перешли из диапазона 0..1 в диапазон 0..2. Результат наложения текстуры будет зависеть от установленного для данного текстурного объекта режима адресации текселей. Для режима повторения (GL_REPEAT) результат приведен ниже:

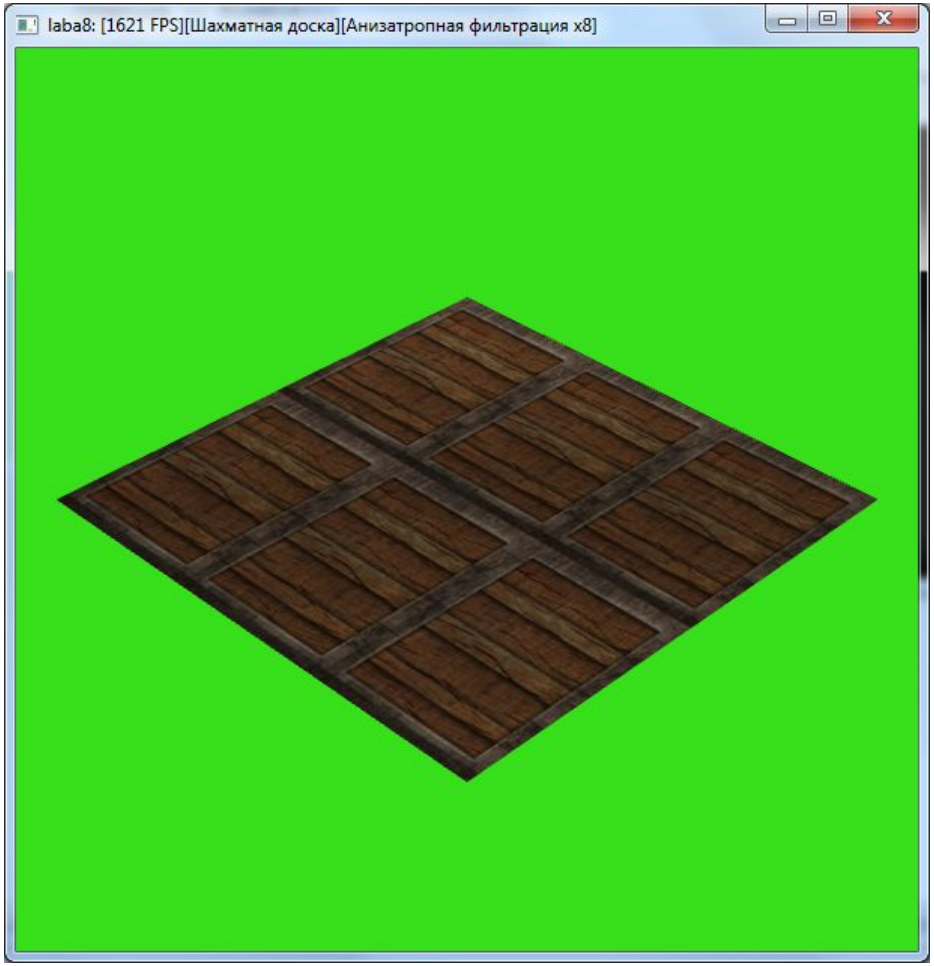




Использование матриц текстурных координат.

Подобная матрица, в которой вместо единиц на главной диагонали стоят другие значения, называется матрицей масштабирования. Значения в главной диагонали задают масштабирование по каждой оси. Пример наложения текстуры для матрицы представленной ниже:

$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Использование матриц текстурных координат.

Процесс установки матриц текстурных координат выполняется по следующему сценарию:

1. Выбор текстурного блока для которого будет задаваться матрица текстурный координат:

```
// выбор активного текстурного блока
glActiveTexture (GL_TEXTURE0);
```

2. Выбор текущей матрицы. OpenGL позволяет работать с матрицами нескольких типов, например матрицей проекции (GL_PROJECTION), матрицей наблюдения моделей (GL_MODELVIEW) и матрицей текстурных координат (GL_TEXTURE). Перед модификацией матрицы необходимо указать какую именно матрицу мы будет изменять:

```
// выбор текущей матрицы
glMatrixMode(GL_TEXTURE);
```

3. Модификация матрицы текстурных координат ранее выбранного текстурного блока. С текстурными матрицами можно выполнять все те же действия, что и с матрицами наблюдения модели: загружать матрицу, домножать матрицу но новую матрицу, загружать единичную матрицу, помещать матрицу в стек и извлекать матрицу из стека. Отдельно следует отметить, что каждый текстурный блок обладает собственным стеком текстурных матриц.

4. После модификации текстурной матрицы, для того чтобы избежать возможных ошибок. Лучше вернуться к матрице наблюдения модели:

```
// возврат к матрице наблюдения модели
glMatrixMode(GL_MODELVIEW);
```

Использование матриц текстурных координат.

Пример загрузки матрицы текстурных координат для нулевого текстурного блока:

```
// выбор активного текстурного блока
glActiveTexture (GL_TEXTURE0);
// выбор текущей матрицы
glMatrixMode (GL_TEXTURE);
// загрузка матрицы масштабирования
Glfloat TM[] = {      2, 0, 0, 0,      // нулевой столбец
           0, 2, 0, 0,      // первый столбец
           0, 0, 2, 0,      // второй столбец
           0, 0, 0, 1};     // третий столбец
glLoadMatrixf (TM);
// возврат к матрице наблюдения модели
glMatrixMode (GL_MODELVIEW);
```

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Следует еще раз отметить следующие основные моменты преобразования текстурных координат:

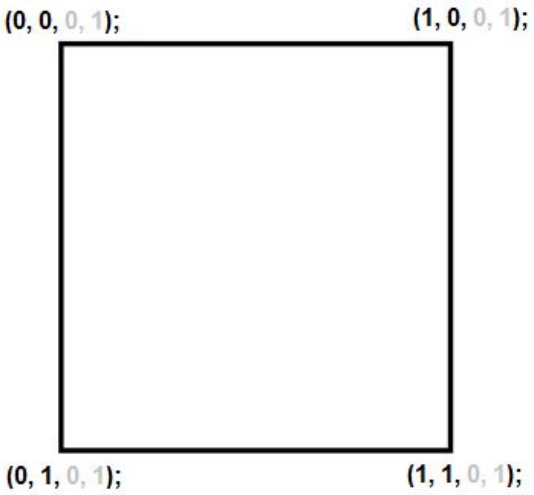
1. Текстурные координаты всегда имеют внутреннее представление (s, t, r, q). Если некоторые компоненты не заданы, они принимают значение по умолчанию (0, 0, 0, 1).
2. Текстурная координата вершины, переданная для определенного блока, всегда умножается на матрицу текстурных координат этого блока.
3. Для наложения двухмерных текстур из полученного вектора (s, t, r, q) берутся два компонента (s, t) которые используются для обращения к текстурному объекту.
4. Текстурный объект по переданным текстурным координатам (s, t) и с учетом режима адресации текселей и режима фильтрации выбирает некоторое значение из массива текселей и возвращает его в текстурный блок для наложения на фрагмент.

Использование матриц текстурных координат.

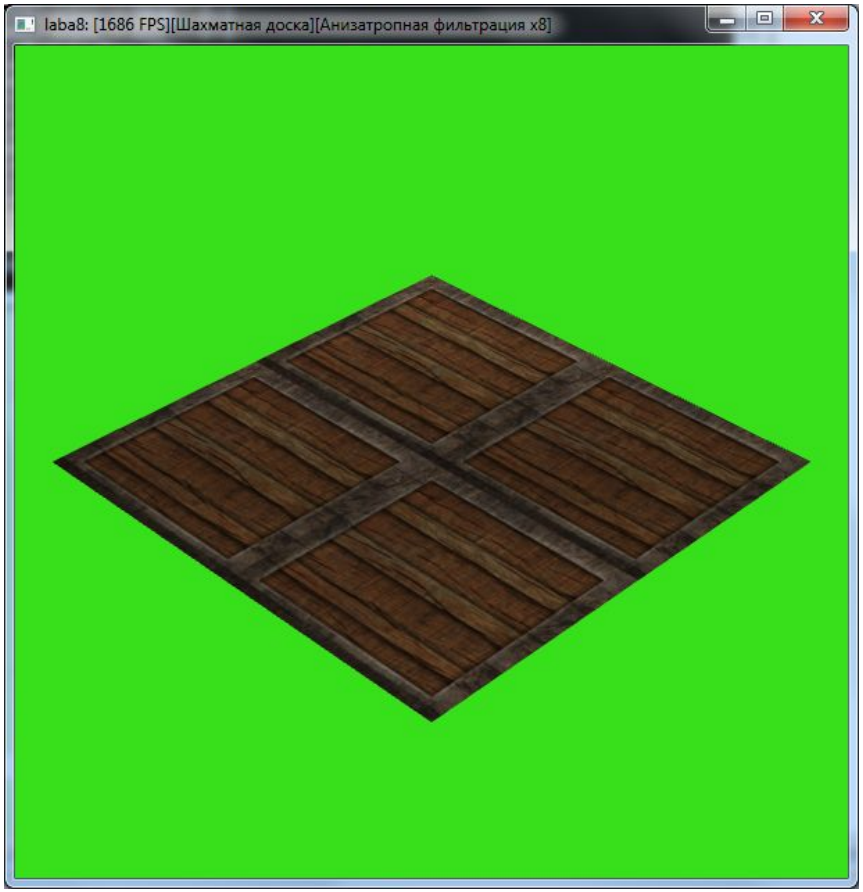
Используя матрицу преобразования текстурных координат, можно добиться различных эффектов – сдвиг, масштабирование или поворот изображения текстуры, накладываемой на полигон, без непосредственного изменения текстурных координат хранимых в буфере VBO.

Существуют стандартные матрицы преобразования, например:

1) Матрица масштабирования:

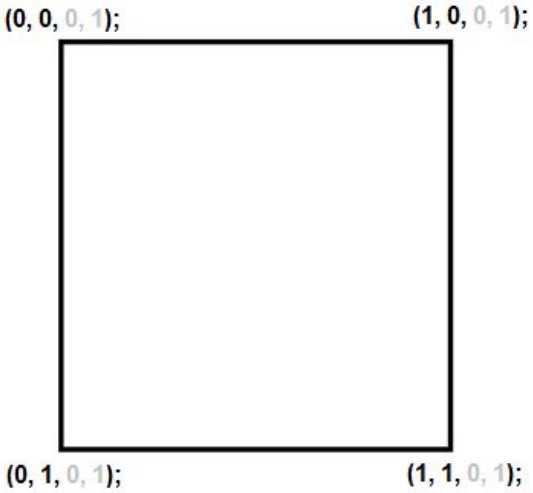


$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ p \\ q \end{bmatrix}$$

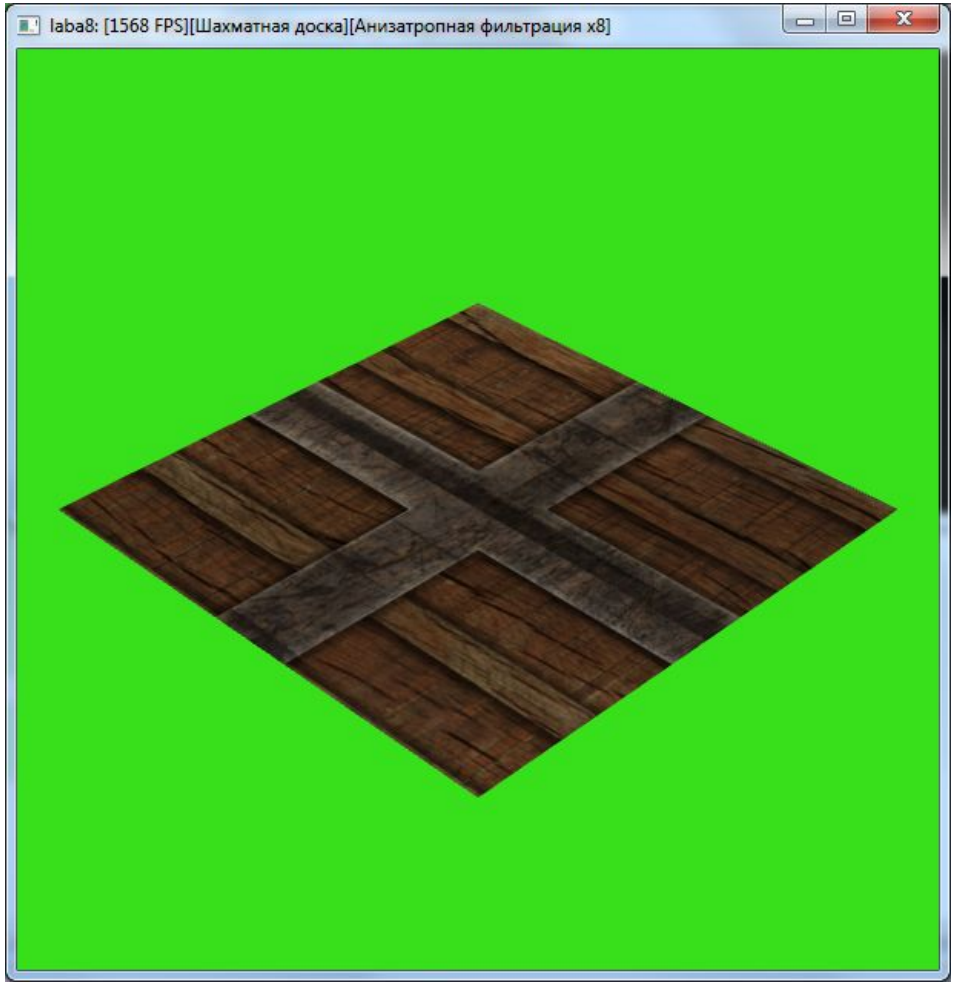


Использование матриц текстурных координат.

2) Матрица переноса:

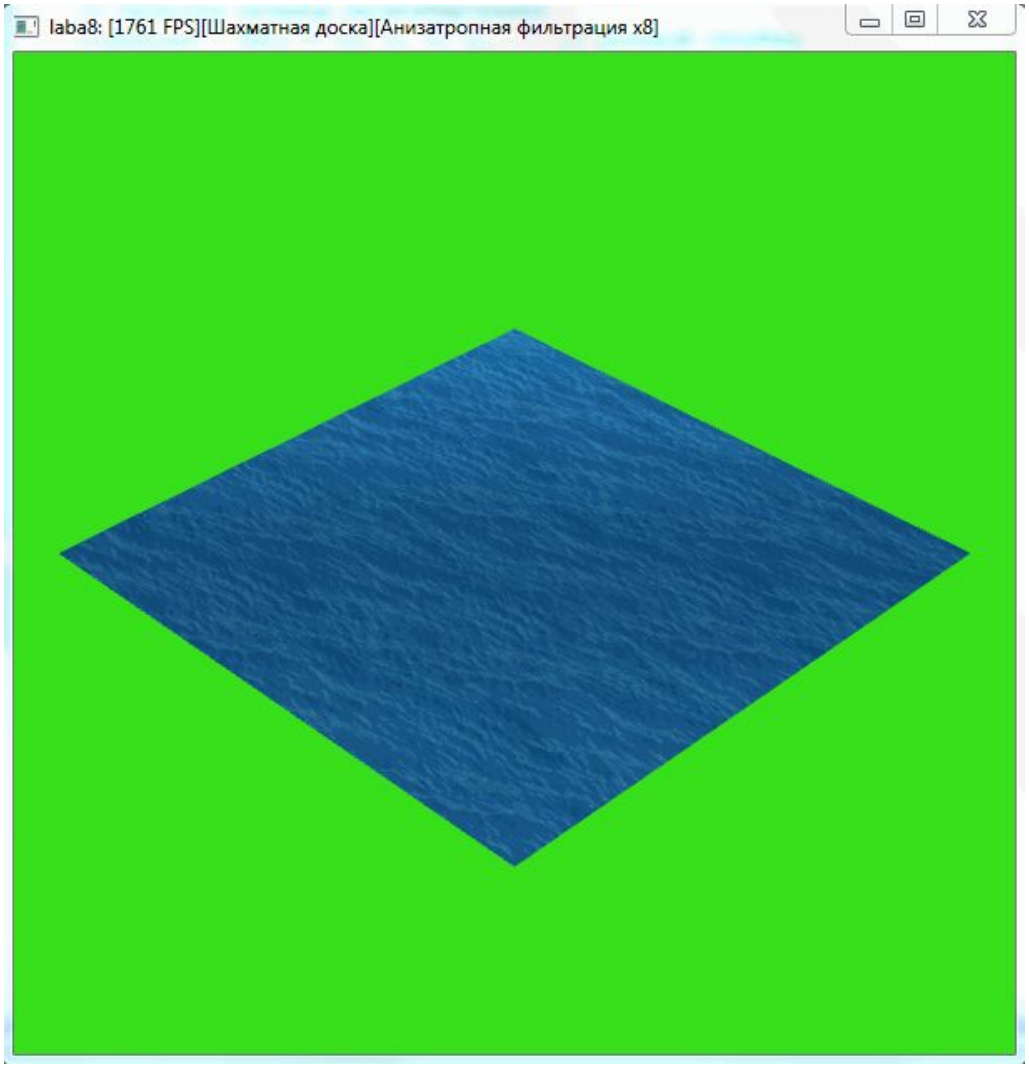


$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ p \\ q \end{bmatrix}$$



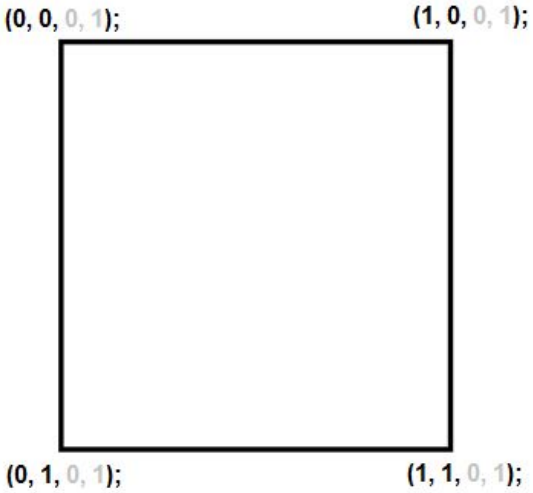
Использование матриц текстурных координат.

При постоянном плавном изменении матрицы переноса можно добиться интересных эффектов движущейся поверхности:

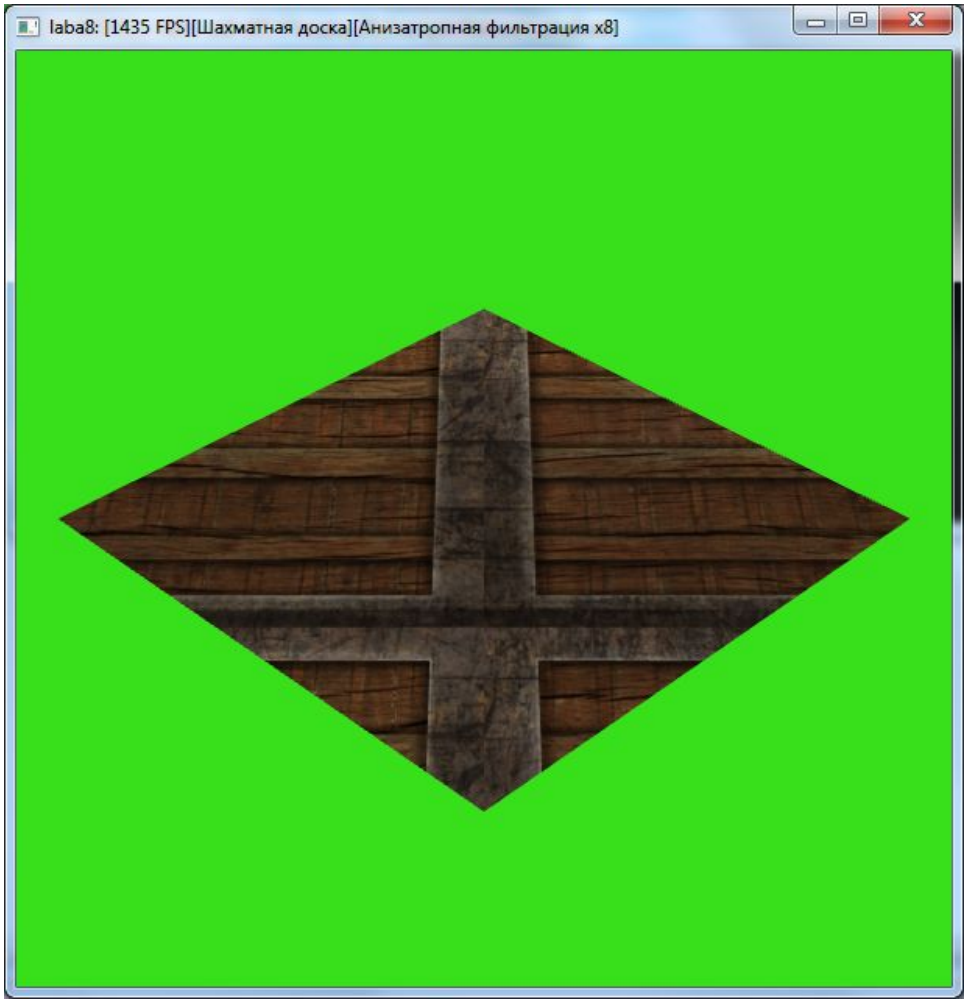


Использование матриц текстурных координат.

3) Матрица поворота на уголь альфа вокруг оси OZ:



$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 & 0 \\ \sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ p \\ q \end{bmatrix}$$

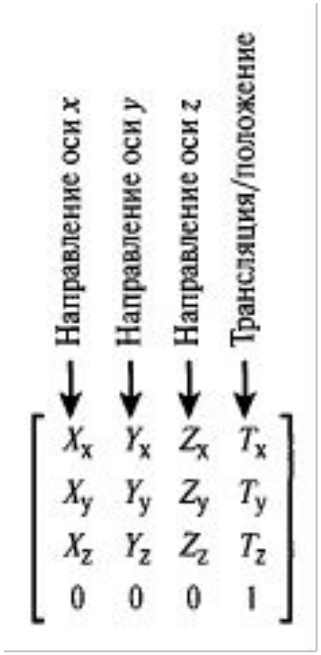


Использование матриц текстурных координат.

4) Общий случай перехода к новой системе координат.

Так же можно напрямую вычислить матрицу перехода из одной системы координат в другую стандартным способом. Для такого перехода, из старой системы координат в новую, необходимо выразить оси старой системы координат через координаты новой системы координат, а так же выразить центр старой системы координат, через координаты новой системы координат.

Полученные вектора используются для построения матрицы переходи из старой системы координат в новую. Умножив координаты из старой системы координат на данную матрицу, мы получим координаты выраженные в новой системе координат:



X – вектор описывающий вектор Ox старой системы координат в координатах новой системы координат, то есть вектор, определяющий какие координаты имеет единичный вектор Ox, если рассматривать его с позиции новой системы координат.

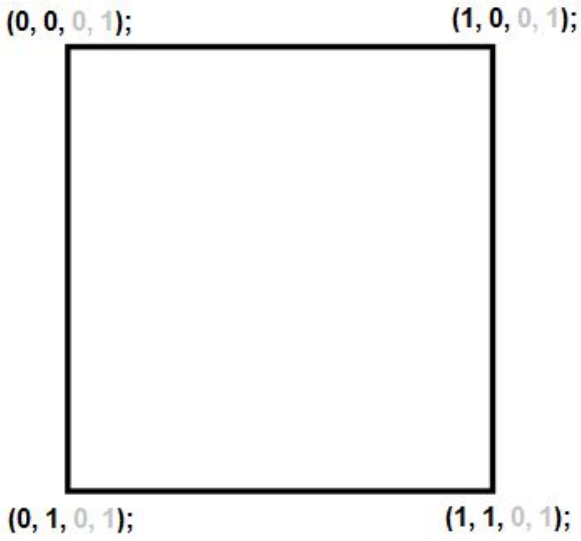
Y - вектор описывающий вектор Oy старой системы координат в координатах новой системы координат.

Z - вектор описывающий вектор Oz старой системы координат в координатах новой системы координат.

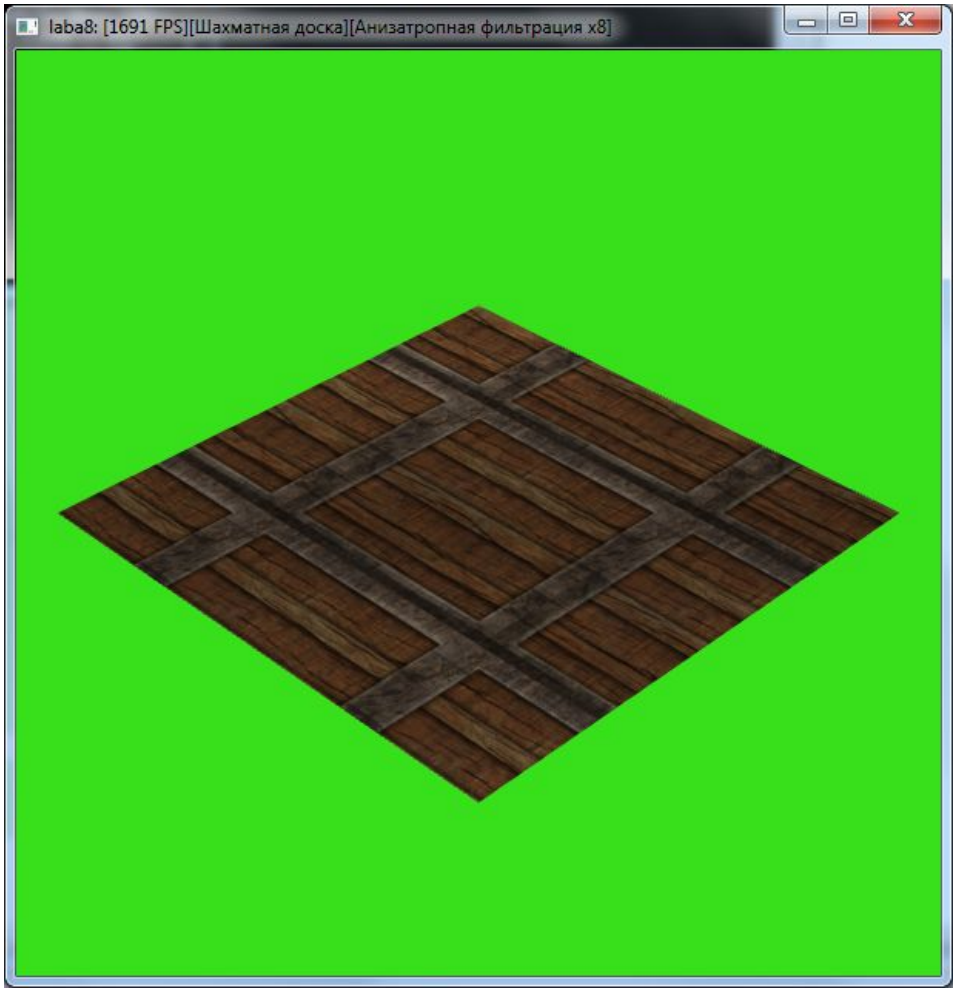
T – вектор описывающий начало старой системы координат в координатах новой системы координат.

Использование матриц текстурных координат.

Рассмотрим пример из лабораторной работы. Для полигона, чьи текстурные координаты заданы в соответствии с рисунком ниже, необходимо создать такую текстурную матрицу, чтобы рисунок текстуры был уменьшен в два раза и выводился по центру:

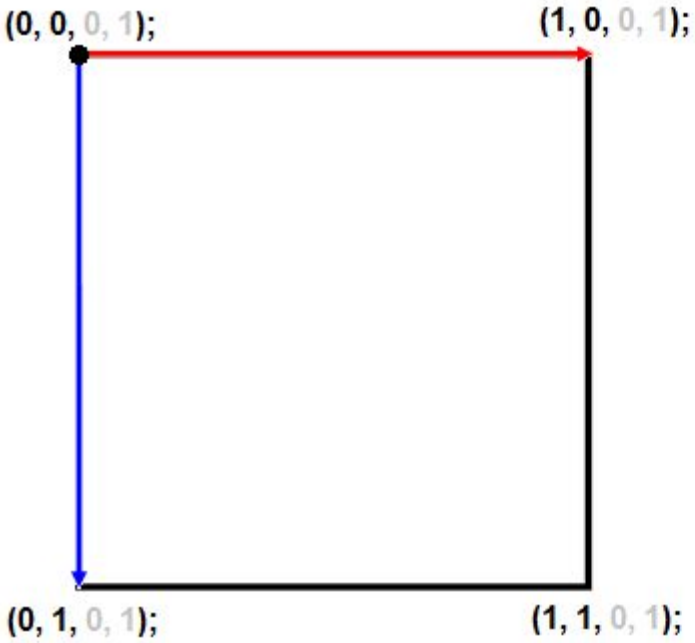


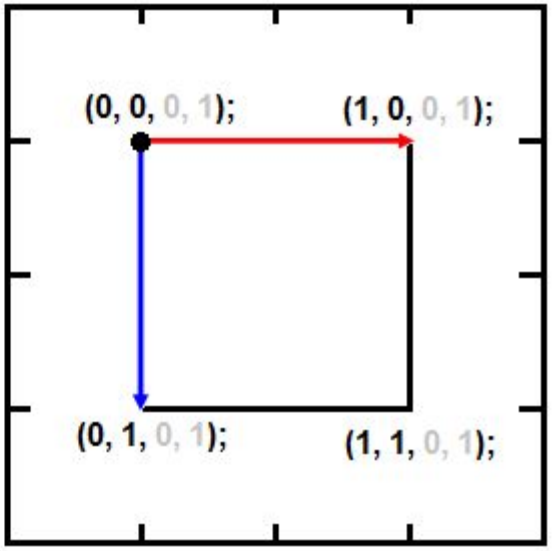
$$\begin{bmatrix} TM \end{bmatrix}$$



Использование матриц текстурных координат.

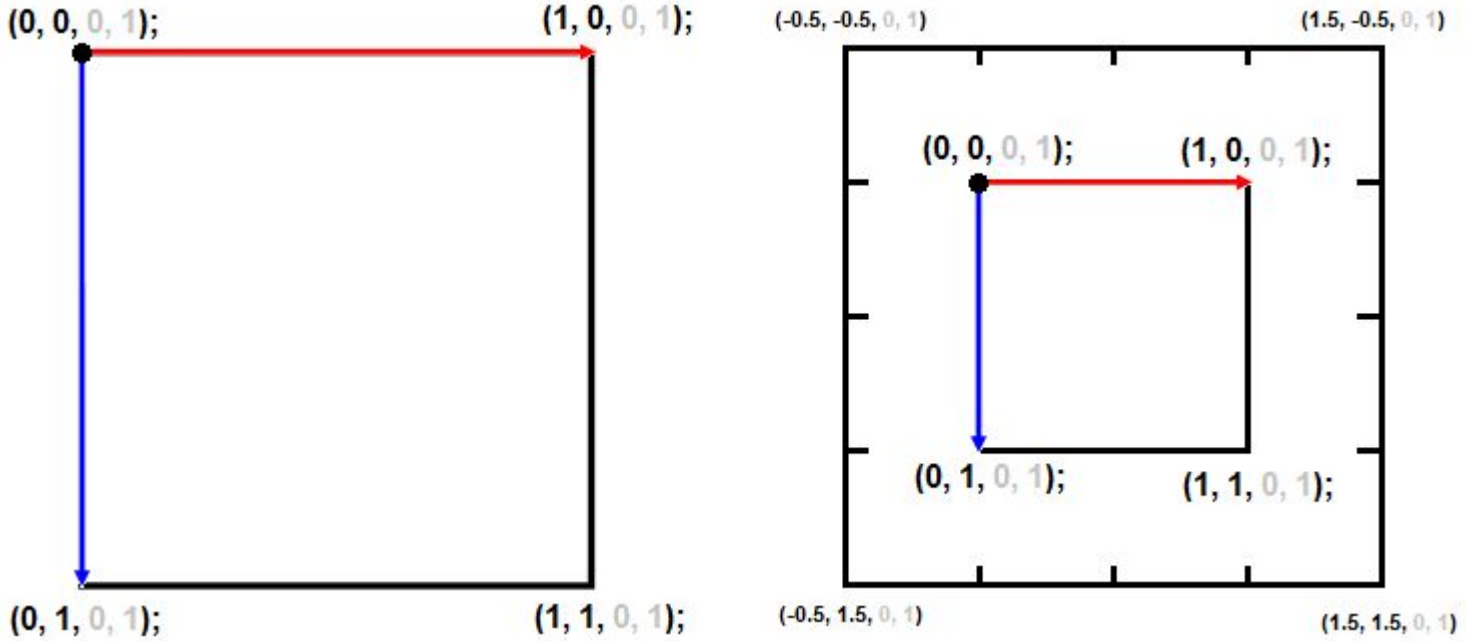
Таким образом, нам необходимо преобразовать систему координат представленную слева, где текстурные координаты меняются от нуля до одного, в систему координат представленную справа, где система координат построена так, что в центральной части текстурные координаты меняются от нуля до одного, а по бокам выходят из этого диапазона:





Использование матриц текстурных координат.

Определим, чему должны быть равны текстурные координаты в вершинах полигона и выразим оси старой системы координат, через оси новой системы координат:



Таким образом, оси старой системы координат будут равны:

X = (2, 0, 0);

Y = (0, 2, 0);

Z = (0, 0, 1);

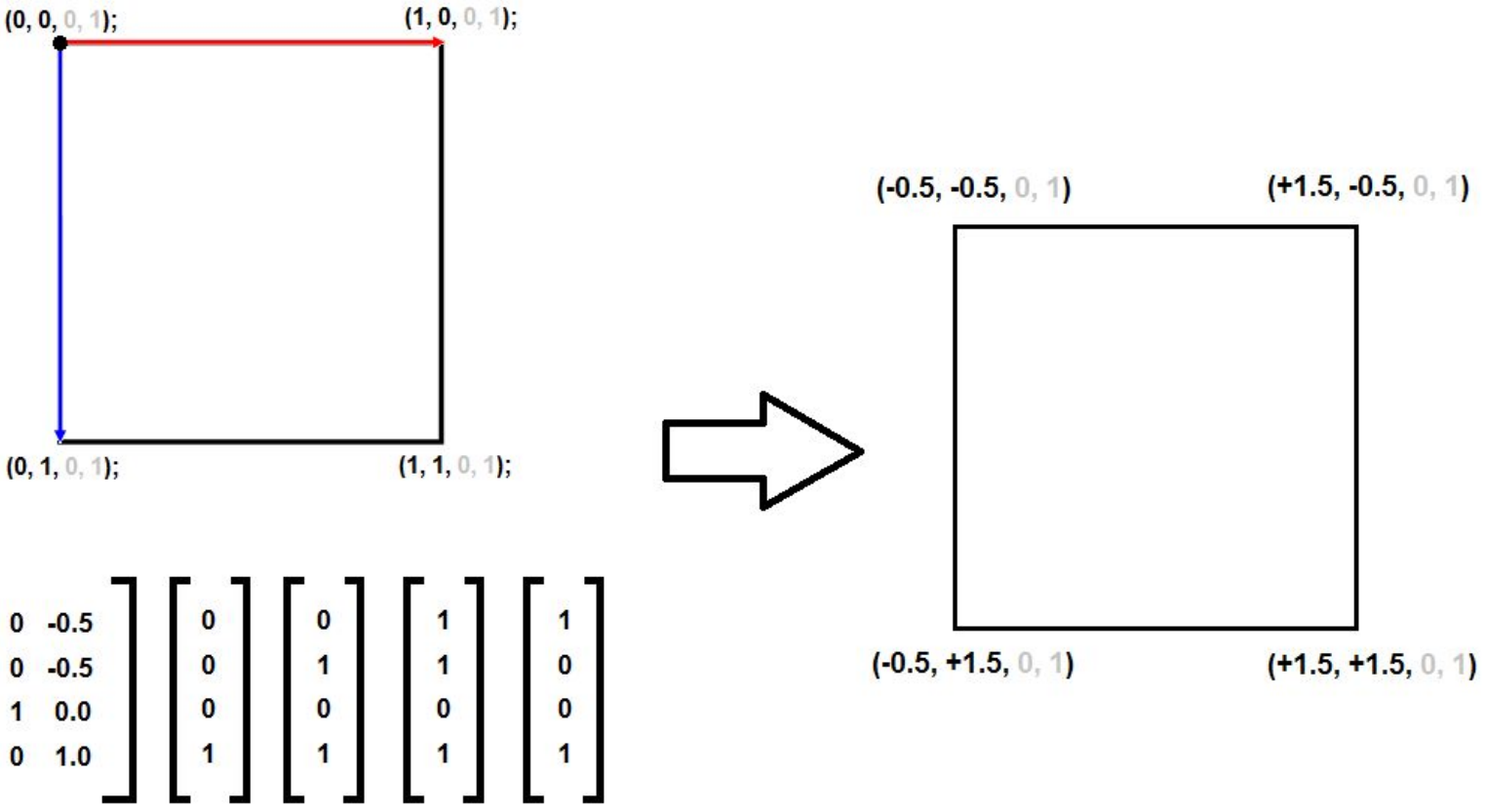
$$\begin{bmatrix} 2 & 0 & 0 & -0.5 \\ 0 & 2 & 0 & -0.5 \\ 0 & 0 & 1 & 0.0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

Начало старой системы координат будет равно:

T = (-0.5, -0.5, 0)

Использование матриц текстурных координат.

Посмотрим как ведут себя текстурные координаты, умноженные на эту матрицу:

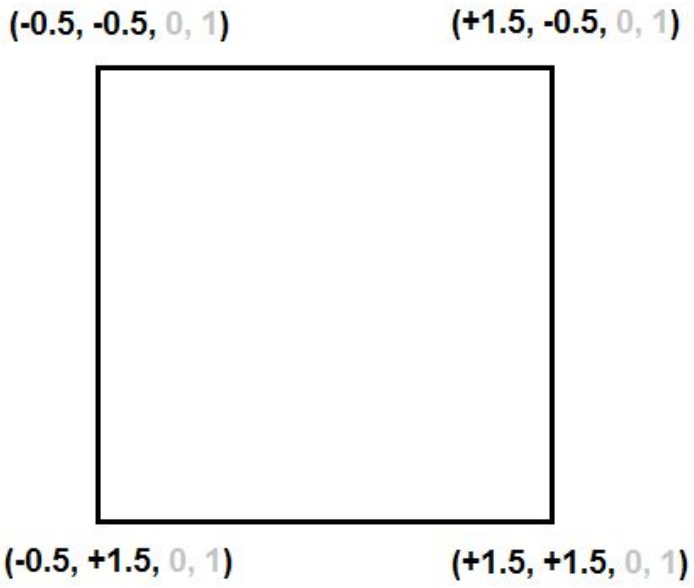


$$\begin{bmatrix} 2 & 0 & 0 & -0.5 \\ 0 & 2 & 0 & -0.5 \\ 0 & 0 & 1 & 0.0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Режим адресации текселей (wrap mode).

После того, как в некотором текстурном модуле текстурные координаты были умножены на текстурную матрицу происходит обращение к текстурному объекту. Для двухмерной текстуры текстурному объекту передаются текстурные координаты (s, t) по которым ищется значение в текстуре.

При этом возможна ситуация, когда текстурные координаты выходят из диапазона (0, 1) и необходимо принять решение, какой тексель должен быть выбран в этом случае. За это отвечает специальный режим, который называется режимом адресации текселей.



Режим адресации текселей (wrap mode).

Режима адресации текселей, так же как и любые другие параметры текстурного объекта, устанавливаются с помощью функции **glTexParameter*.**

Для каждой текстурной координаты (s, t или r) можно установить свой собственный режим адресации текселей. В качестве имени параметра используются следующие константы:

GL_TEXTURE_WRAP_S – режим адресации текселей для текстурной координаты S.

GL_TEXTURE_WRAP_T – режим адресации текселей для текстурной координаты T.

GL_TEXTURE_WRAP_R – режим адресации текселей для текстурной координаты R.

Каждый из этих параметров может принимать следующие значения:

GL_REPEAT – режим повторения;

GL_MIRRORED_REPEAT – режим повторения с отражением;

GL_CLAMP_TO_EDGE – отсечение по крайнему ряду текселей в текстуре;

GL_CLAMP_TO_BORDER – отсечение по границе.

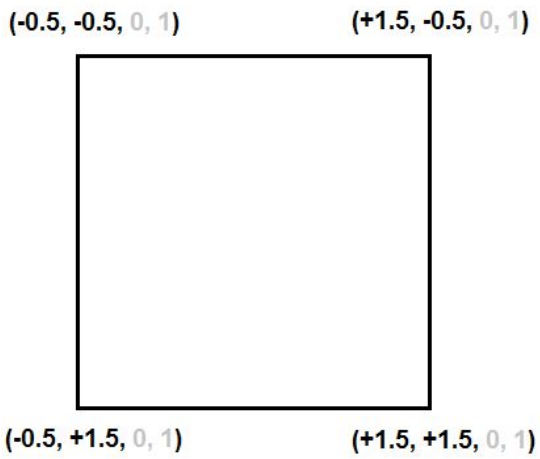
Пример задания режима адресации текселей:

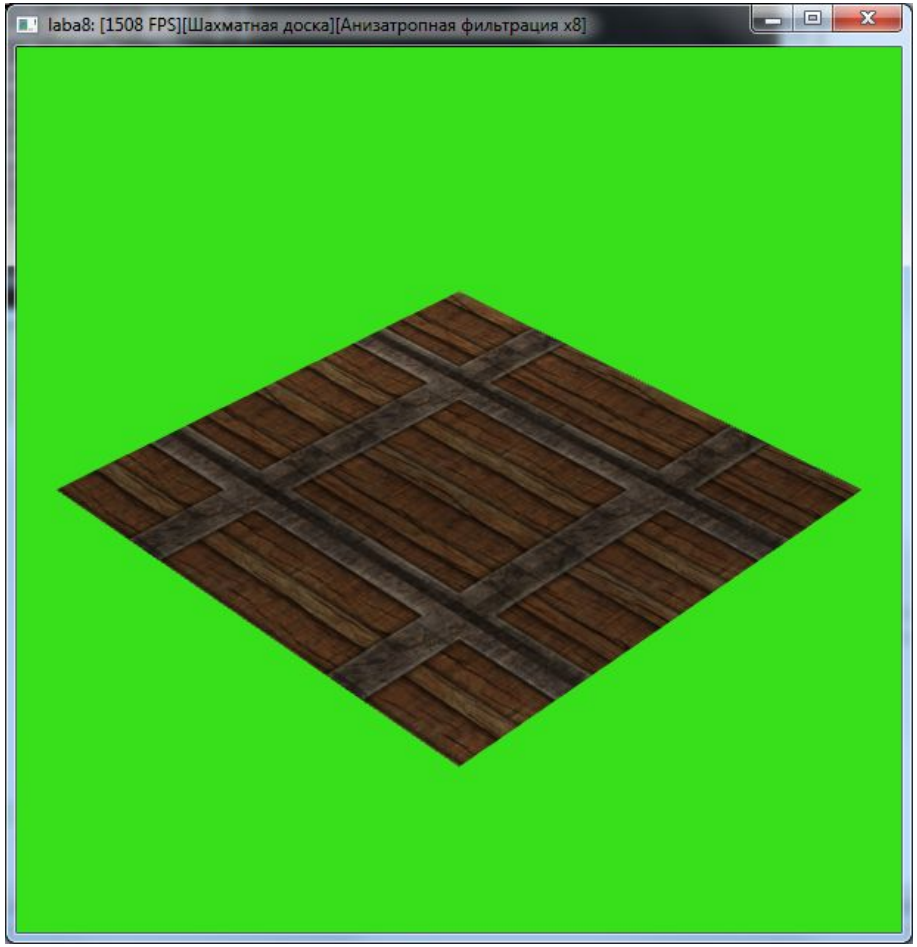
```
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

При задании режима адресации следует помнить о контексте, в частности режим адресации текселей устанавливается для того текстурного объекта, который связан с активным текстурным блоком.

Режим адресации текселей (wrap mode).

Рассмотрим режим **GL_REPEAT** для полигона с текстурными координатами представленными на рисунке ниже. В этом случае целая часть текстурной координаты игнорируется и берется только дробная часть:



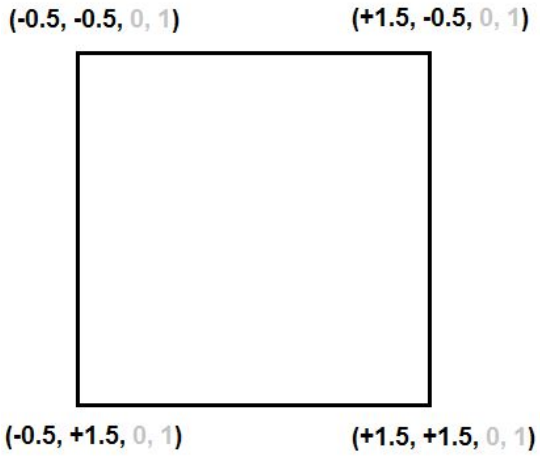


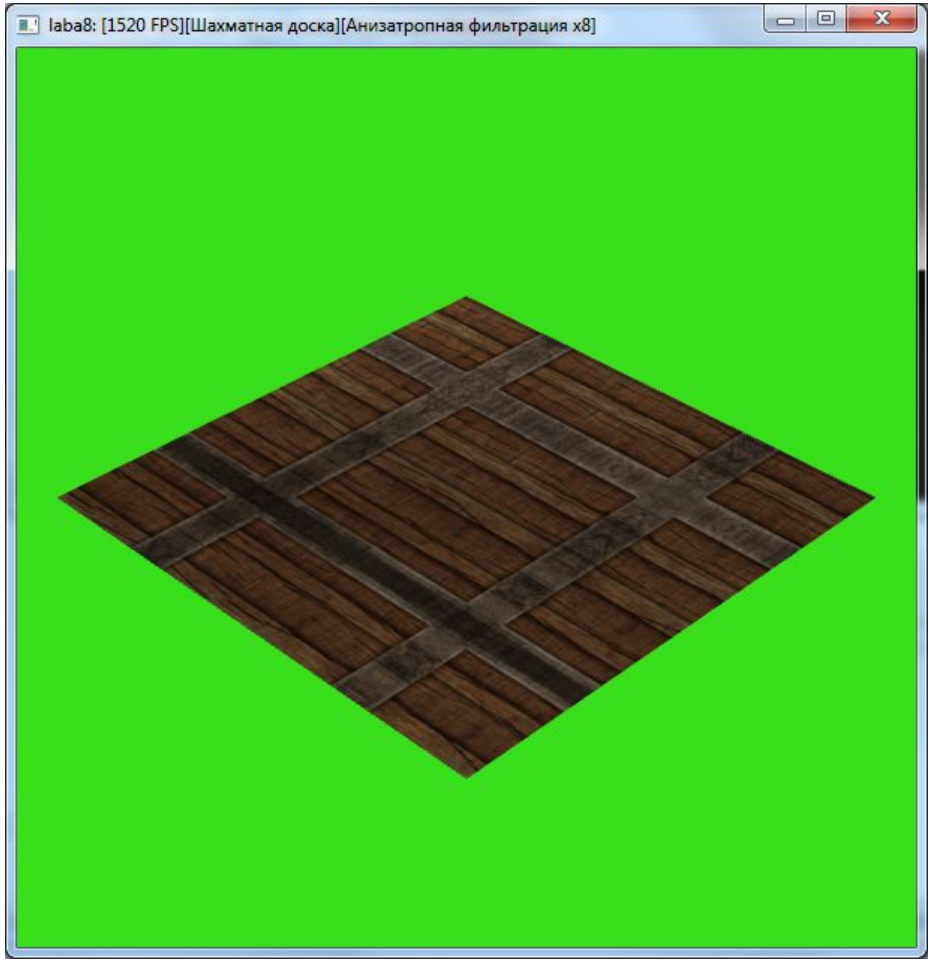
Режим адресации текселей (wrap mode).

Режим **GL_MIRRORED_REPEAT** определен следующим образом:

1) Если целая часть текстурной координат четная, то берется дробная часть текстурной координаты;
2) Если целая часть текстурной координаты нечетная, то берется значение (1 – дробная часть)

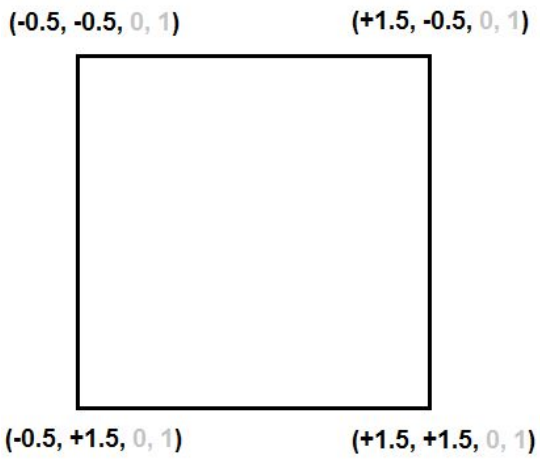
Логически это эквивалентно зеркальному отражению на стыке двух текстур.

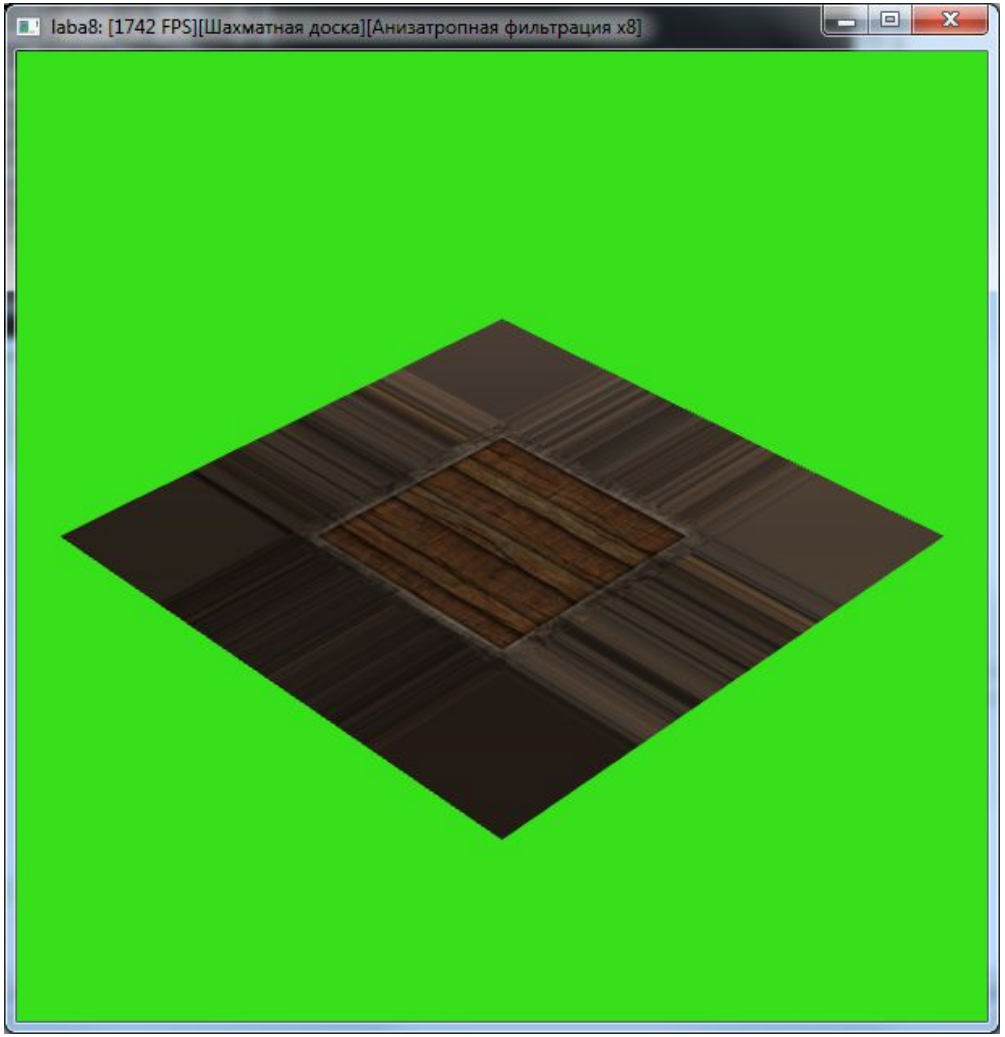




Режим адресации текселей (wrap mode).

Режим **GL_CLAMP_TO_EDGE** отсекает текстурные координаты до диапазона (0..1). Все значения большие единице становятся равными единице, все значения меньшие нуля становятся равными нулю:





Режим адресации текселей (wrap mode).

Режим **GL_CLAMP_TO_BORDER** подразумевает что с текстурным объектом связанно такое понятие как граница, цвет которой является параметром текстурного объекта. В случае выхода текстурной координаты за пределы диапазона (0..1) вместо значения из текстуры возвращается цвет границы.

Для задания цвета границы используется функция **glTexParameterfv** где в качестве имени параметра указывается константа GL_TEXTURE_BORDER_COLOR. Значением параметра является вектор (r, g, b, a) который задает цвет и, что особенно важно, степень прозрачности границы.

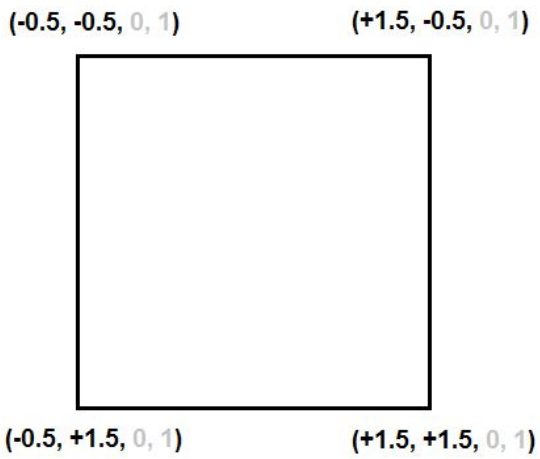
Пример установки цвета границы приведен ниже:

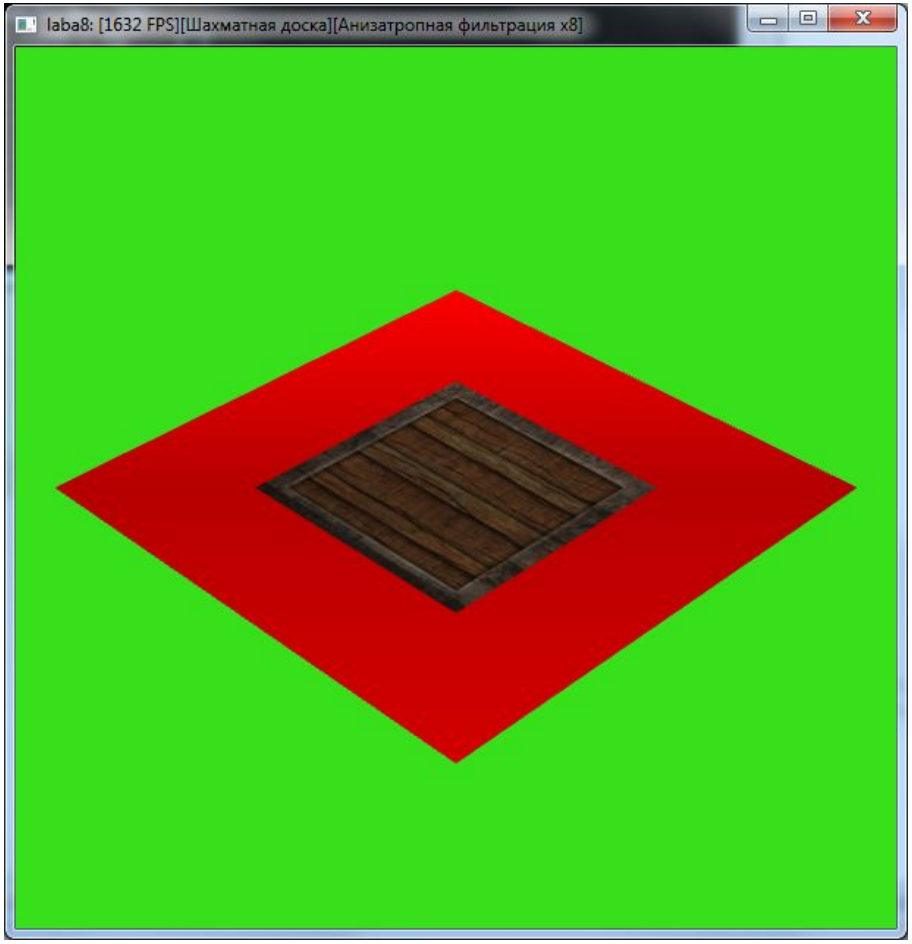
```
float bcolor[] = {1.0, 0.0, 0.0, 0.2};
glTexParameterfv (GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, bcolor);
```

Значение по умолчанию для цвета границы – (0, 0, 0, 0).

Режим адресации текселей (wrap mode).

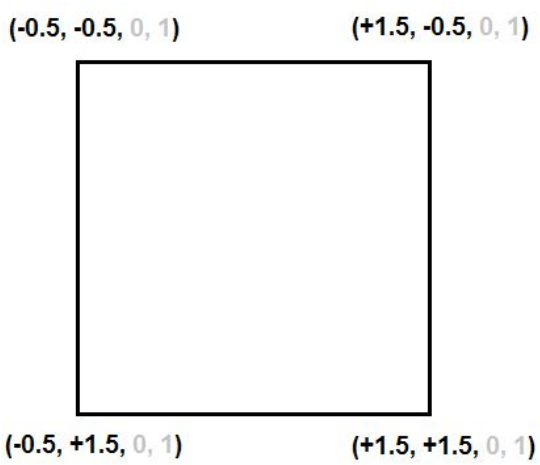
Результат применения режима GL_CLAMP_TO_EDGE. Для фрагментов, чьи текстурные координаты выходят из диапазона (0..1) значение полученное из текстуры равно цвету границы:

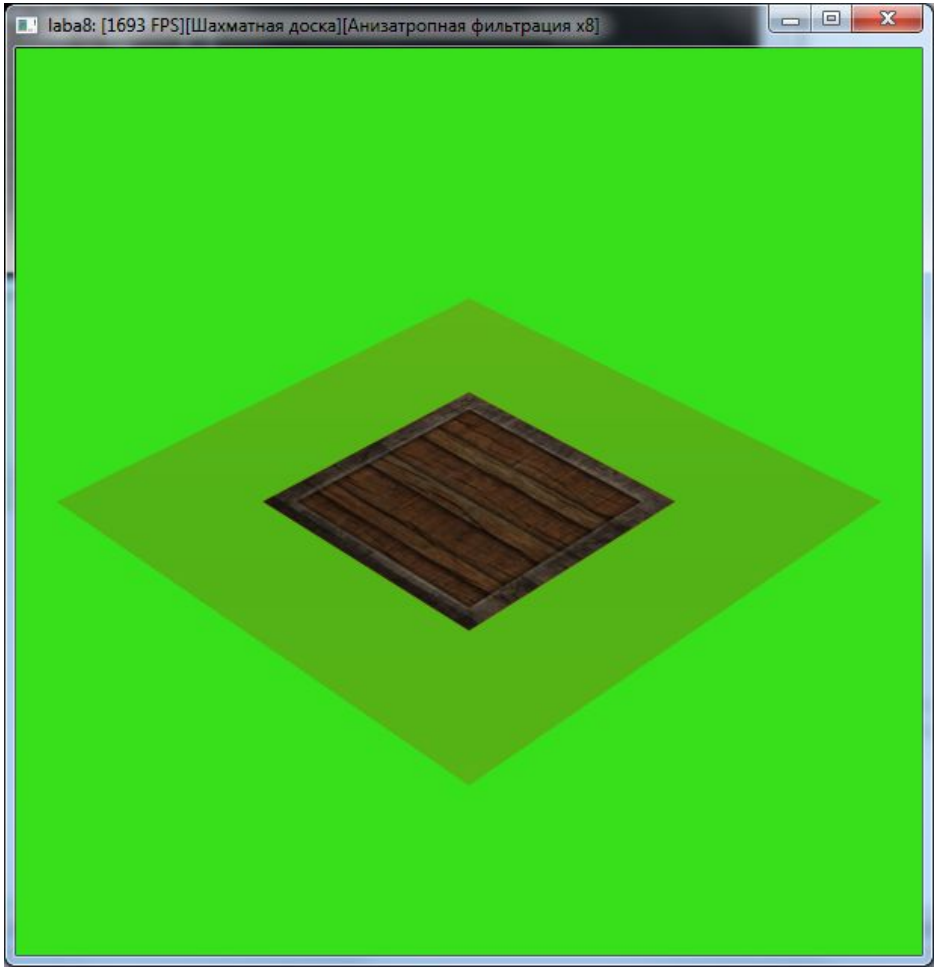




Режим адресации текселей (wrap mode).

Однако, в случае использования альфа-значения цвета границы (например - использования режима цветового наложения) можно получить интересный эффект:





Режим наложения текстуры.

После того, как для фрагмента по его текстурным координатам было выбрано значение из текстуры, это значение можно каким-либо образом использовать для вычисления итогового цвета.

За это отвечает режим наложения текстур – параметр, который задается для каждого текстурного блока отдельно. Задание режима наложения текстур выполняется с помощью функции void glTexEnv*. Функция имеет следующий прототип (один из вариантов):

```
void glTexEnvi ( GLenum   target,
      GLenum   pname,
      GLint    param);
```

Первый параметр **target** – задает для какой группы параметров будут задаваться настройки. Возможные значения:

GL_TEXTURE_ENV – группа параметров, определяющих режим наложения текстур, то есть то, как значение из текстуры будет комбинироваться с ранее вычисленным цветом фрагмента.

GL_TEXTURE_FILTER_CONTROL - группа, содержащая параметры дополнительных настроек фильтрации. Используется довольно редко.

GL_POINT_SPRITE – группа параметров, отвечающих за вывод спрайтов – специальный режим, который нами не рассматривается, поэтому параметры данной группы не модифицируются.

Режим наложения текстуры.

Второй параметр **pname** – задает имя устанавливаемого параметра. Список возможных значений зависит от того, какая группа параметров была указанно в первом параметре функции. Для группы GL_TEXTURE_ENV возможными значениями могут быть:

GL_TEXTURE_ENV_MODE – режим наложения текстур;

GL_TEXTURE_ENV_COLOR – цвет текстурного блока (дополнительный параметр который может участвовать в наложении текстуры);

Так же существуют дополнительные параметры, применяемые для режима наложения GL_COMBINE, такие как:

GL_COMBINE_RGB

GL_COMBINE_ALPHA

GL_SRC0_RGB, GL_SRC1_RGB, GL_SRC2_RGB

GL_SRC0_ALPHA, GL_SRC1_ALPHA, GL_SRC2_ALPHA

GL_OPERAND0_RGB, GL_OPERAND1_RGB, GL_OPERAND2_RGB

GL_OPERAND0_ALPHA, GL_OPERAND1_ALPHA, GL_OPERAND2_ALPHA

GL_RGB_SCALE, GL_ALPHA_SCALE

Режим наложения текстуры.

Третий параметр **param** – задает значение параметра. Для различных параметров существует различные возможные значения.

Наиболее значимым параметром текстурного блока является режим наложения текстур (параметр GL_TEXTURE_ENV_MODE), который может принимать следующие значения: GL_ADD, GL_MODULATE, GL_DECAL, GL_BLEND, GL_REPLACE, or GL_COMBINE.

Таким образом для установки режима наложения текстуры необходимо выполнить следующую последовательность действий:

```
glActiveTexture (GL_TEXTURE0);
glBindTexture (GL_TEXTURE_2D,TexIndex);
glEnable (GL_TEXTURE_2D);
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

В данном случае для нулевого текстурного блока была выбрана текстура (привязан текстурный объект), разрешено наложение двухмерной текстуры и выбран режим наложения текстуры (GL_MODULATE).

Далее будут подробно рассмотрены различные режимы наложения текстуры.

Режим наложения текстуры GL_REPLACE:

Режим наложения **GL_REPLACE** игнорирует ранее вычисленный цвет и в качестве итогового цвета данного блока использует значение полученное из текстуры:

GL_REPLACE	Внутренний формат текстуры RGB	Внутренний формат текстуры RGBA
Цвет (C)	$C = C_t$	$C = C_t$
Альфа (A)	$A = A_p$	$A = A_t$

Индексы: t – значение полученное из связанного текстурного объекта;

p – ранее вычисленного значение. Для нулевого блока - цвет самого фрагмента, для первого блока – цвет полученный после нулевого текстурного блока и т.д.

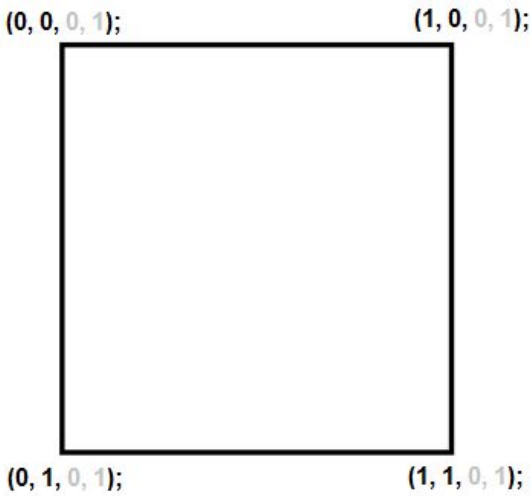
Поскольку ранее вычисленный цвет не учитывается то:

1) При выводе объектов использующих данный режим наложения текстуры можно отключать режим расчета освещения, для экономии времени, поскольку вычисленное значение не будет использоваться.
2) Использование данного режима наложения в текстурных блоках отличных от нулевого неоправданно, поскольку так же будет учитываться только цвет полученный в этом текстурном блоке, а ранее вычисленные цвета будут игнорироваться.

Данный режим часто используется для вывода элементов интерфейса, например курсора мышки, кнопок, иконок и прочего. В случае, если нам надо наложить текстуру на объект для которого нам важно знать его освещенность, требуется использовать другие режимы.

Режим наложения текстуры GL_REPLACE:

Пример наложения текстуры с режимом наложения GL_REPLACE:



Режим наложения текстуры GL_MODULATE :

Режим наложения **GL_MODULATE** покомпонентно перемножает ранее вычисленный цвет и цвет полученный из текстуры:

GL_MODULATE	Внутренний формат текстуры RGB	Внутренний формат текстуры RGBA
Цвет (C)	$C = C_p * C_t$	$C = C_p * C_t$
Альфа (A)	$A = A_p$	$A = A_p * A_t$

Индексы: t – значение полученное из связанного текстурного объекта;

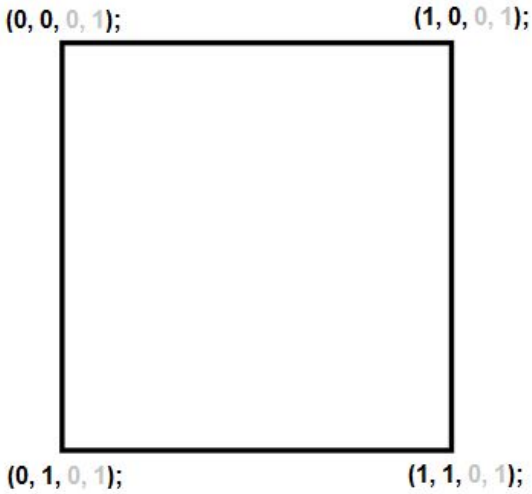
p – ранее вычисленного значение. Для нулевого блока - цвет самого фрагмента, для первого блока – цвет полученный после нулевого текстурного блока и т.д.

В результате применения данного режима наложения учитывается как ранее вычисленный цвет, так и значение из текстуры. То есть темные участки остаются темнее, светлые светлее, но с учетом индивидуального цвета фрагментов определяемых текстурой.

Режим **GL_MODULATE** может учитываться не только в нулевом текстурном блоке, где текстура комбинируется с ранее вычисленным цветом фрагмента, но и в последующим блоках, например для наложения детальной текстуры.

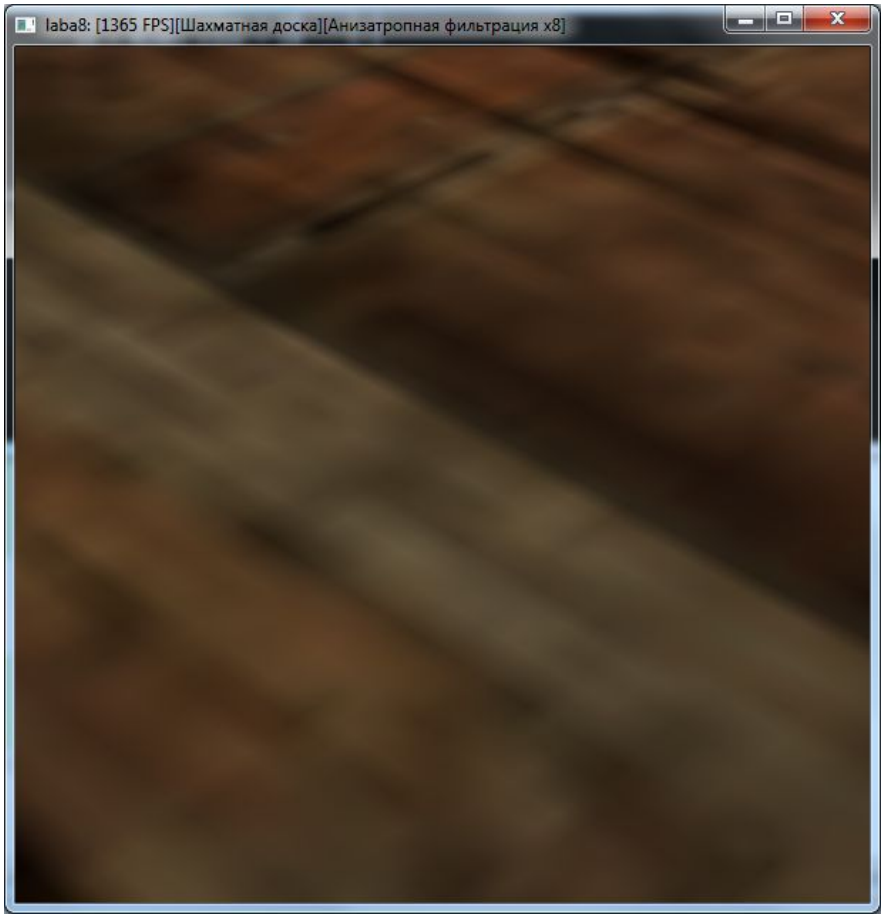
Режим наложения текстуры GL_MODULATE :

Пример наложения текстуры с режимом GL_MODULATE :



Режим наложения текстуры GL_MODULATE :

Если накладывать только одну текстуру, то при приближении к объекту заметно размытие текстуры:



Режим наложения текстуры GL_MODULATE :

Для того, чтобы нивелировать этот недостаток, используют текстуру детализации. В этом случае в нулевом блоке накладывают основную текстуру, а в первом - текстуру детализации. В обоих блоках используется режим наложения GL_MODULATE, но для первого текстурного блока матрица трансформации устанавливается так, чтобы текстура накладывалась гораздо чаще.

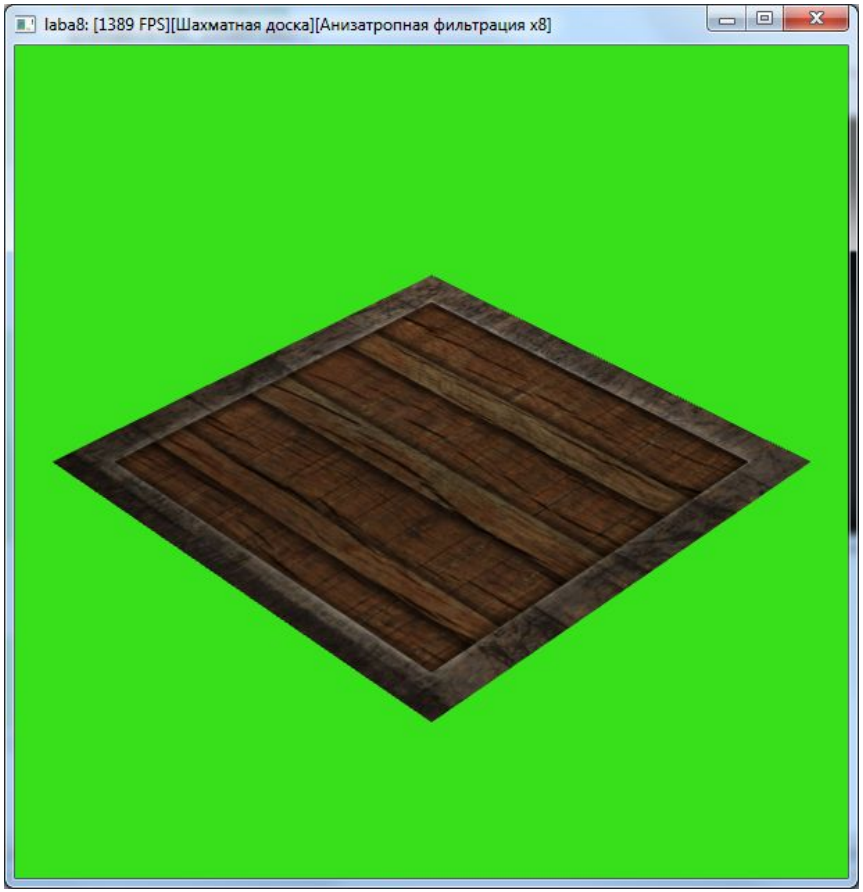
Следует отметить, что итоговая формула расчета цвета будет выглядеть следующим образом:

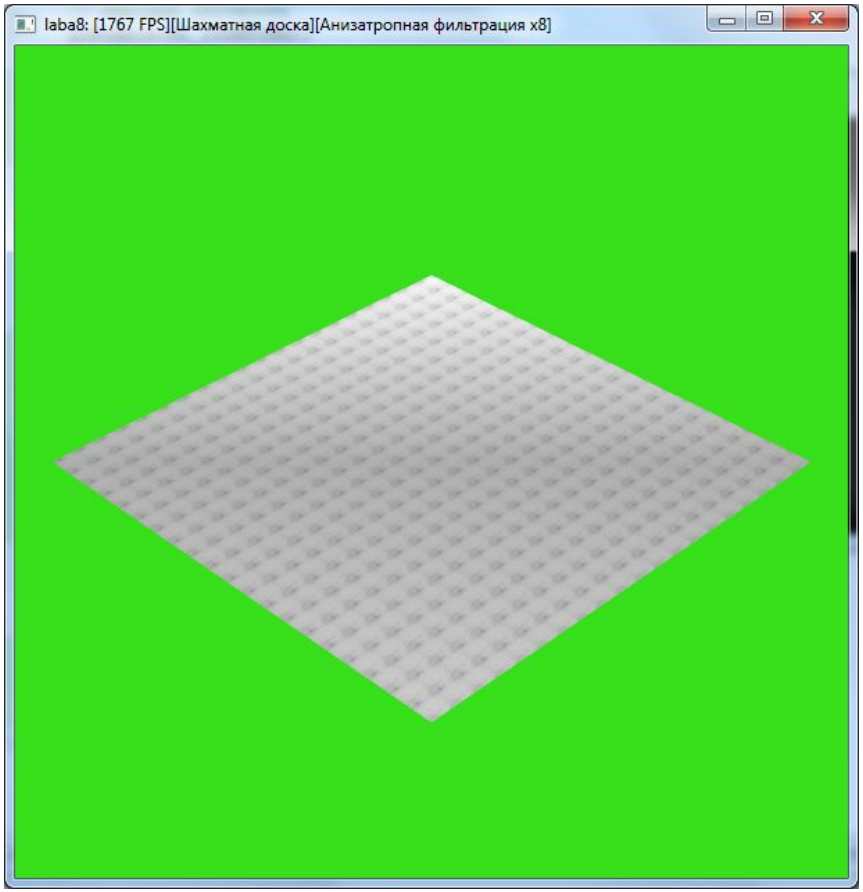
$$C = C_f * C_{t0} * C_{t1}$$

Поскольку компоненты вектора цвета всегда лежать в пределах 0..1, то с каждым таким наложением цвет будет становится все темнее и темнее, поэтому текстура детализации при таком способе задается как можно светлее.

Режим наложения текстуры GL_MODULATE :

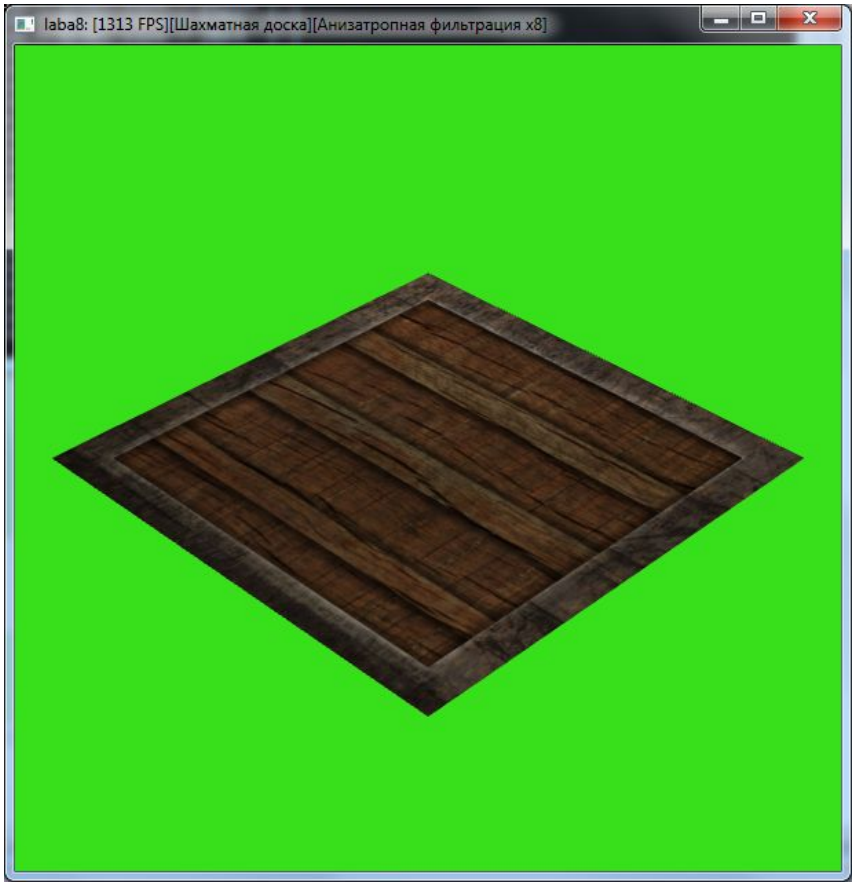
Снизу приведен результат наложения только основной текстуры (слева) и только текстуры детализации (справа). Обратите внимание, что текстура детализации накладывается гораздо чаще.

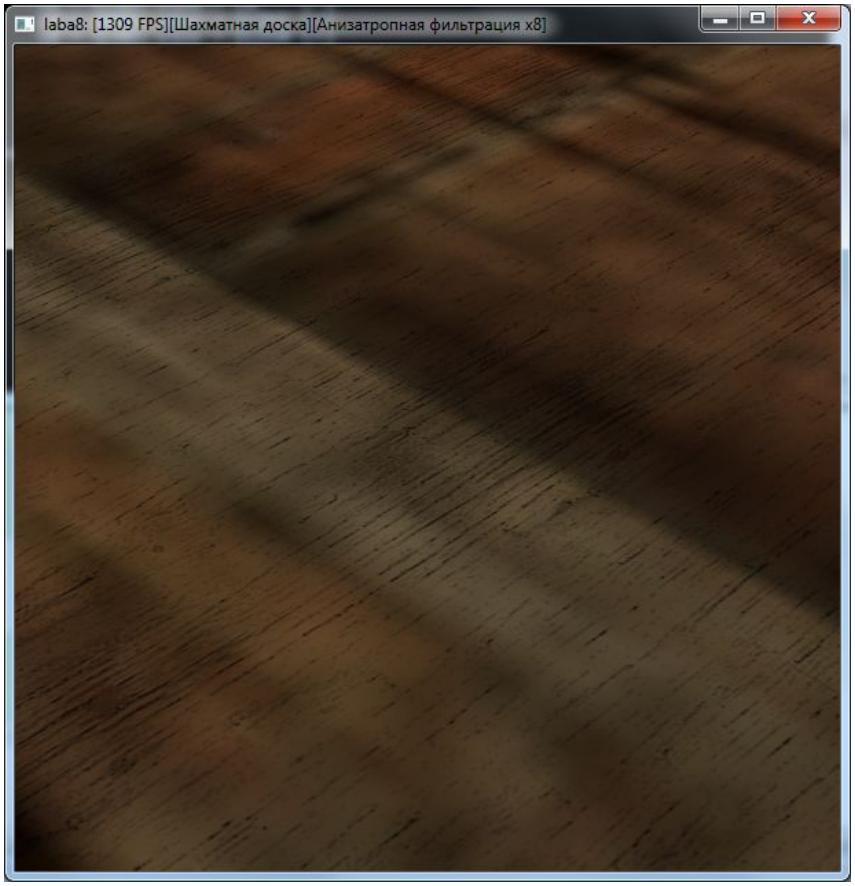




Режим наложения текстуры GL_MODULATE :

Наложение обоих текстур при взгляде издалека и вблизи:





Режим наложения текстуры GL_ADD :

Режим наложения **GL_ADD** покомпонентно складывает ранее вычисленный цвет и цвет полученный из текстуры.

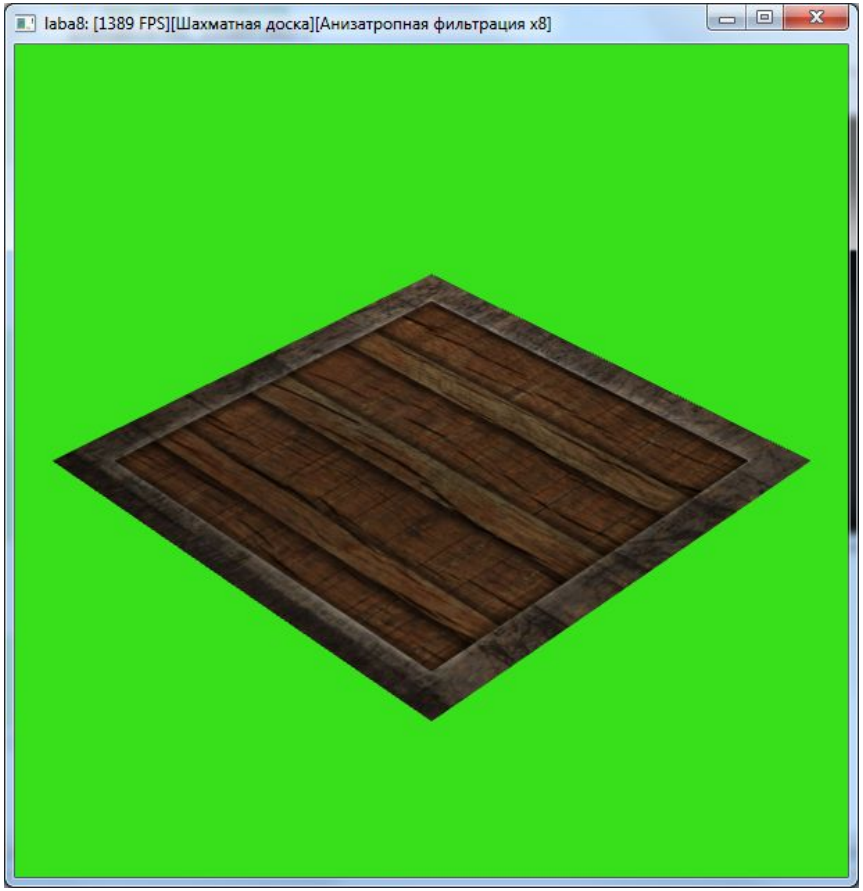
GL_ADD	Внутренний формат текстуры RGB	Внутренний формат текстуры RGBA
Цвет (C)	$C = C_p + C_t$	$C = C_p + C_t$
Альфа (A)	$A = A_p$	$A = A_p * A_t$

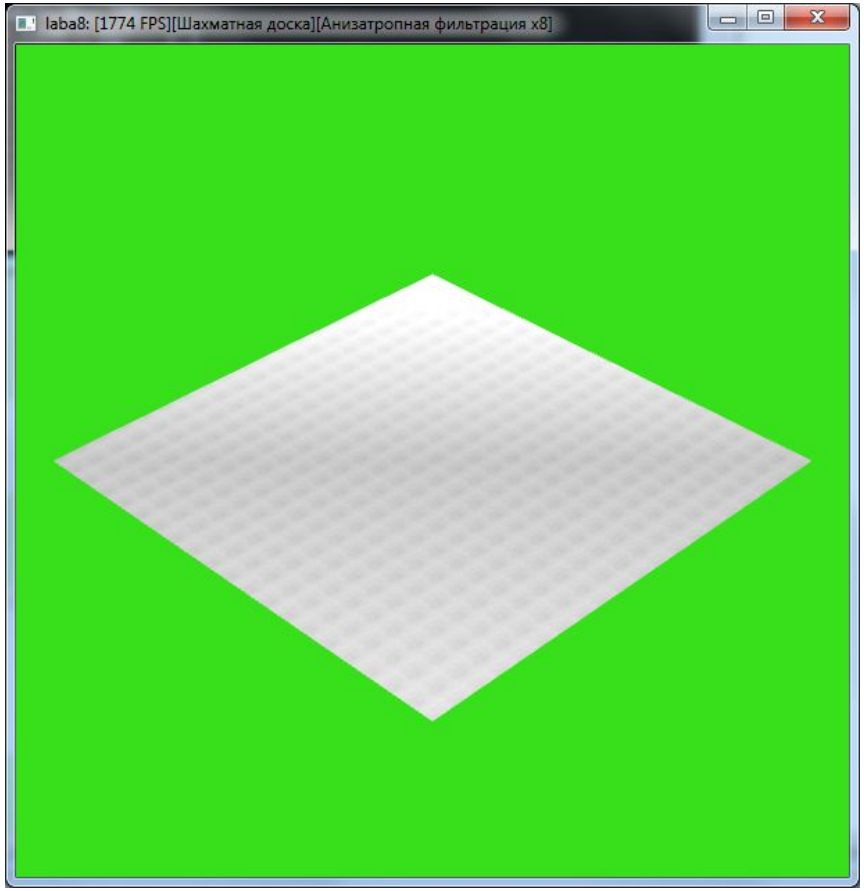
В результате применения данного режима наложения цвета определенных фрагментов осветляются в соответствии со значением из текстуры. Как правило в этих случаях текстура задается в оттенках серого.

Для наложения детальной текстуры вместо режима GL_MODULATE можно использовать режим GL_ADD. Текстура детализации в этом случае задается в оттенках серого и должна быть как можно темнее.

Режим наложения текстуры GL_ADD :

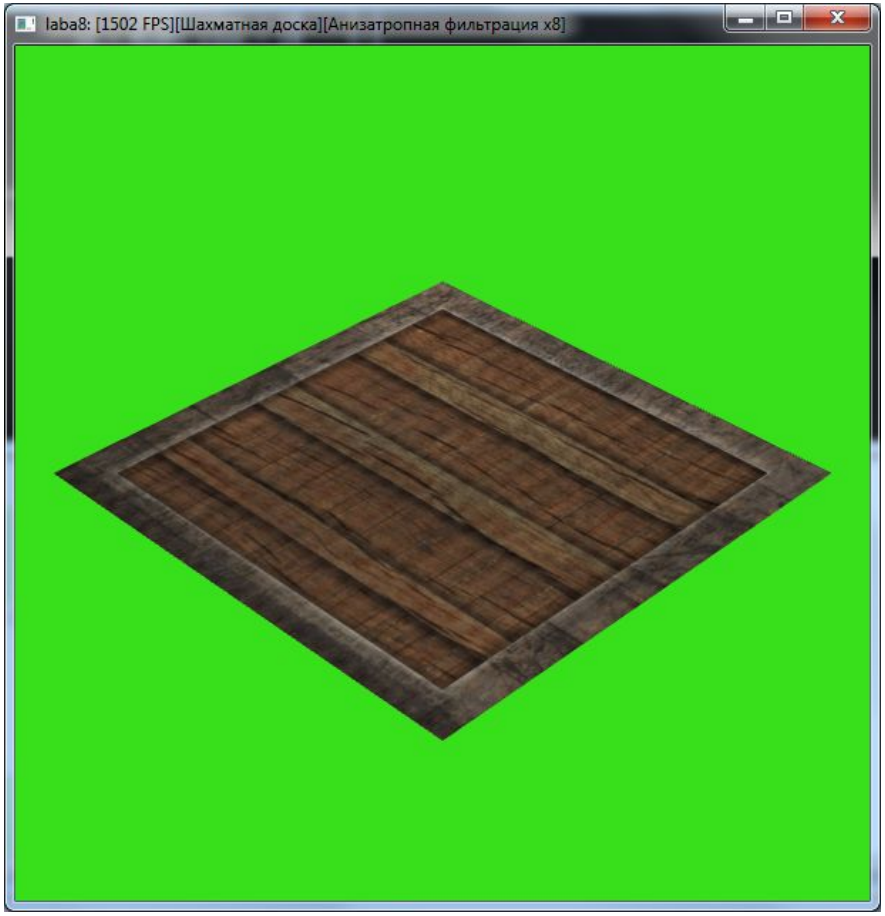
Снизу приведен результат наложения только основной текстуры (слева) и только текстуры детализации (справа). Текстура детализации накладывается с режимом GL_ADD:

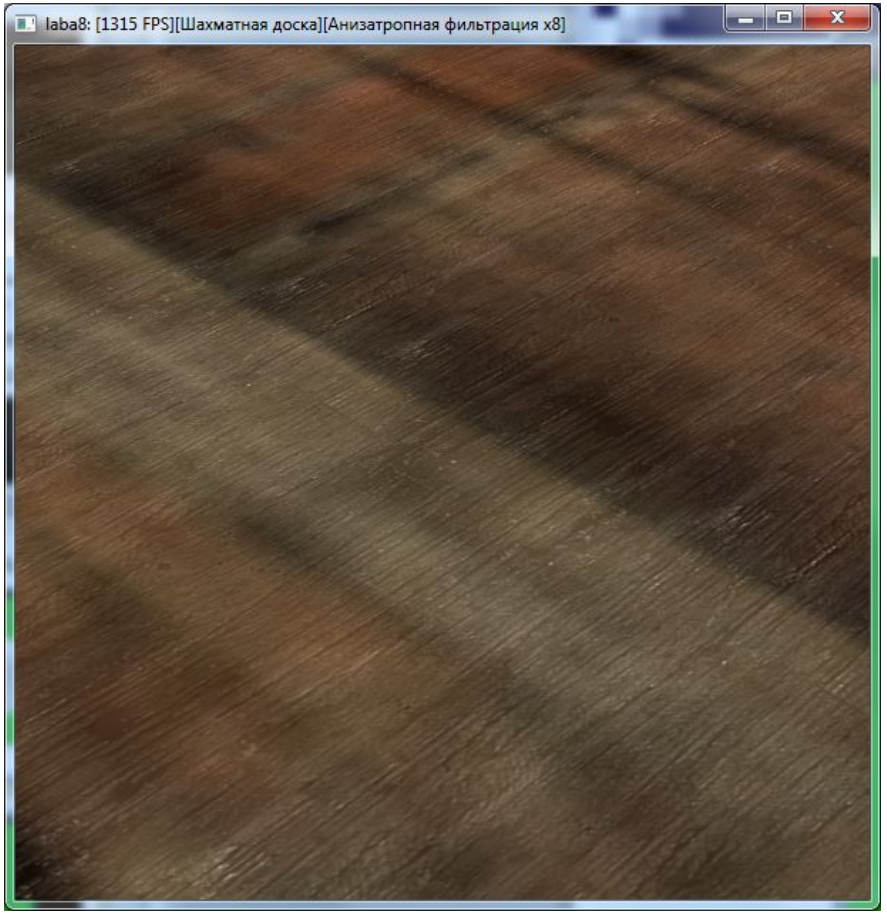




Режим наложения текстуры GL_ADD :

Наложение обоих текстур при взгляде издалека и вблизи:





Режим наложения текстуры GL_DECAL :

Режим наложения **GL_DECAL** позволяет смешивать ранее вычисленное значение со значением из текстуры в пропорции, определяемой альфа-каналом текстуры.

GL_DECAL	Внутренний формат текстуры RGB	Внутренний формат текстуры RGBA
Цвет (C)	$C = C_t$	$C = C_p * (1 - A_t) + C_t * A_t$
Альфа (A)	$A = A_p$	$A = A_p$

Индексы: t – значение полученное из связанного текстурного объекта;

p – ранее вычисленного значение. Для нулевого блока - цвет самого фрагмента, для первого блока – цвет полученный после нулевого текстурного блока и т.д.

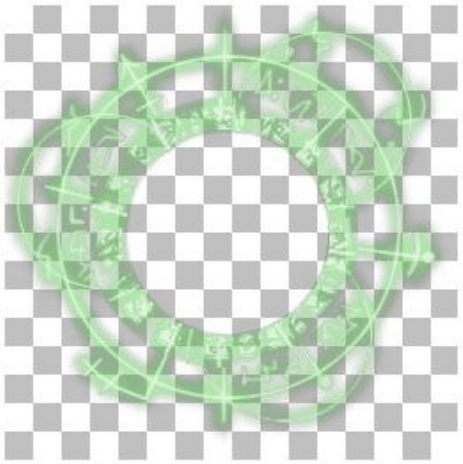
Для режима наложения GL_DECAL внутренний формат всегда должен быть GL_RGBA иначе режим эквивалентен GL_REPLACE. В случае если альфа канал принимает значения ноль или один, то он может задавать либо те места, в которых используется ранее вычисленный цвет (0) либо те, где цвет необходимо заменить на значение из текстуры (1). В последнем случае ранее вычисленный цвет (в том числе освещение) не принимаются во внимание.

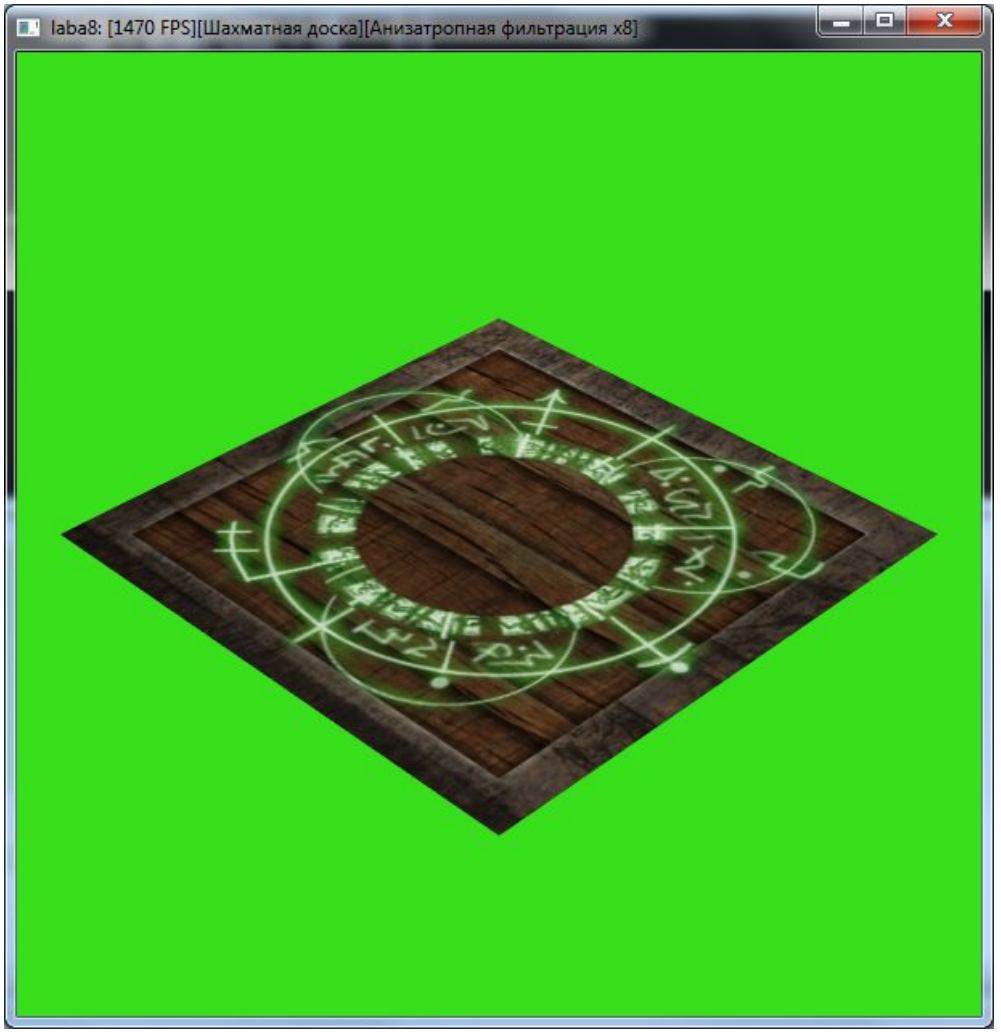
Режим наложения текстуры GL_DECAL :

Рассмотрим результат наложения двух текстур. В нулевом модуле используется режим GL_MODULATE, а в первом – GL_DECAL:

GL_MODULATE

GL_DECAL



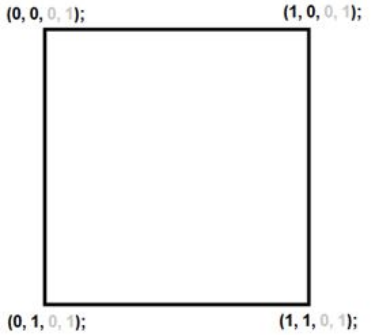


Режим наложения текстуры GL_DECAL :

Так же для задания альфа-канала текстуры накладываемой с режимом GL_DECAL можно использовать цвет границы текстурного объекта совместно с режимом адресации текселей GL_CLAMP_TO_BORDER.

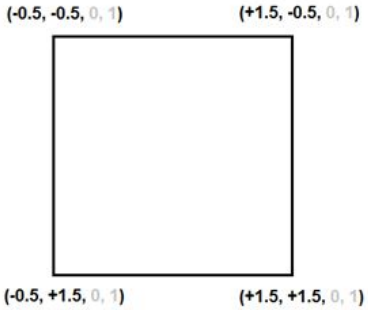
GL_MODULATE

[TM]



GL_DECAL

[TM]



цвет границы = (0, 0, 0, 0)
режим адресации =
GL_CLAMP_TO_BORDER

Режим наложения текстуры GL_DECAL :

Ниже приведен результат наложения второй текстуры с режимом GL_DECAL. Обратите внимание, что там где текстурные координаты выходят за пределы 0..1 используется цвет границы текстурного объекта, который в данном случае равен (0, 0, 0, 0). Поскольку считанное из текстуры значение альфа равно нулю, то для этих фрагментов в соответствии с формулой $C = C_p * (1 - A_t) + C_t * A_t$ итоговым цветом будет ранее вычисленный цвет, а не значение цвет границы или цвет текстуры:

Режим наложения текстуры GL_BLEND :

Режим наложения **GL_BLEND** позволяет смешивать ранее вычисленное значение с фиксированным цветом текстурного блока в пропорции заданной самой текстурой. Смешивание происходит покомпонентно:

GL_BLEND	Внутренний формат текстуры RGB	Внутренний формат текстуры RGBA
Цвет (C)	$C = C_p * (1 - C_t) + C_c * C_t$	$C = C_p * (1 - C_t) + C_c * C_t$
Альфа (A)	$A = A_p$	$A = A_p * A_t$

Индексы: t – значение полученное из связанного текстурного объекта;

p – ранее вычисленного значение. Для нулевого блока цвет самого фрагмента, для первого блока – цвет полученный после нулевого текстурного блока и т.д.

c – константный цвет (цвет текстурного блока)

У каждого текстурного блока есть фиксированный цвет (цвет текстурного блока), который задается с помощью параметра GL_TEXTURE_ENV_COLOR. Пример установки цвета текстурного блока приведен далее:

```
float     EnvColor[] = {1.0, 0.0, 0.0, 1.0};
glTexEnvfv (GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, EnvColor);
```

При режиме наложения **GL_BLEND** в качестве текстуры обычно выступает текстура заданная в оттенках серого (все компоненты одинаковые), которая показывает в какой пропорции следует смешивать ранее вычисленный цвет и цвет текстурного блока. Если значение считанное из текстуры равно (0,0,0) (черный тексель) – будет использоваться ранее вычисленный цвет. Если считан белый тексель (1, 1, 1) – будет использоваться только цвет текстурного блока. Если значение равно (0.5, 0.5, 0.5) то итоговый цвет в равной пропорции учитывает ранее вычисленный цвет и цвет текстурного блока.

Режим наложения текстуры GL_BLEND :

Результат наложения текстуры в режиме GL_BLEND по формуле $C = C_p * (1 - C_t) + C_c * C_t$

GL_MODULATE

GL_BLEND

цвет текстурного блока = (1, 0, 0, 1)

Режим наложения текстуры GL_BLEND :

Преимуществом данного подхода является то, что можно использовать одну и ту же текстуру и, меняя цвет текстурного блока, получить различные эффекты: