Разработка графических приложений

БЕЛЯЕВСКИЙ КИРИЛЛ ОЛЕГОВИЧ, АСП.

KIRILL.BELIAEVSKII@SPBPU.COM

Программа курса

Три лекции:

- Трехмерные модели
- Пространственные преобразования
- Растеризация
- Раскраска и текстурирование
- Z-буферизация и сглаживание

Одна лабораторная:

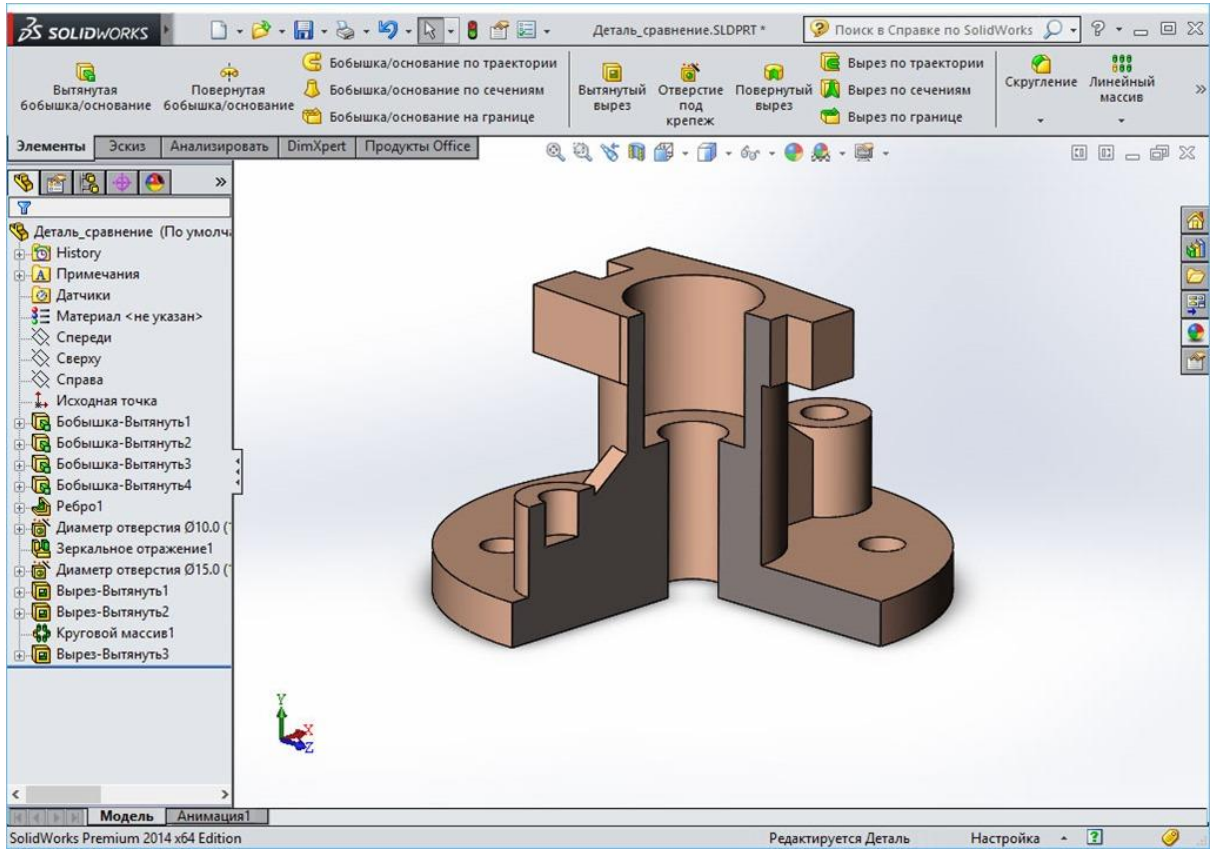
- Растеризация трехмерных моделей на CPU

Твердотельная модель

Результат композиции заданного множества геометрических элементов с применением операций булевой алгебры

Рассматривает тело как сплошную среду, а не как пустой объем, ограниченный поверхностями

Может применяться при анализе сложных конструкций, расчетах прочности и теплопередачи



Воксельная модель

Подкласс твердотельных моделей

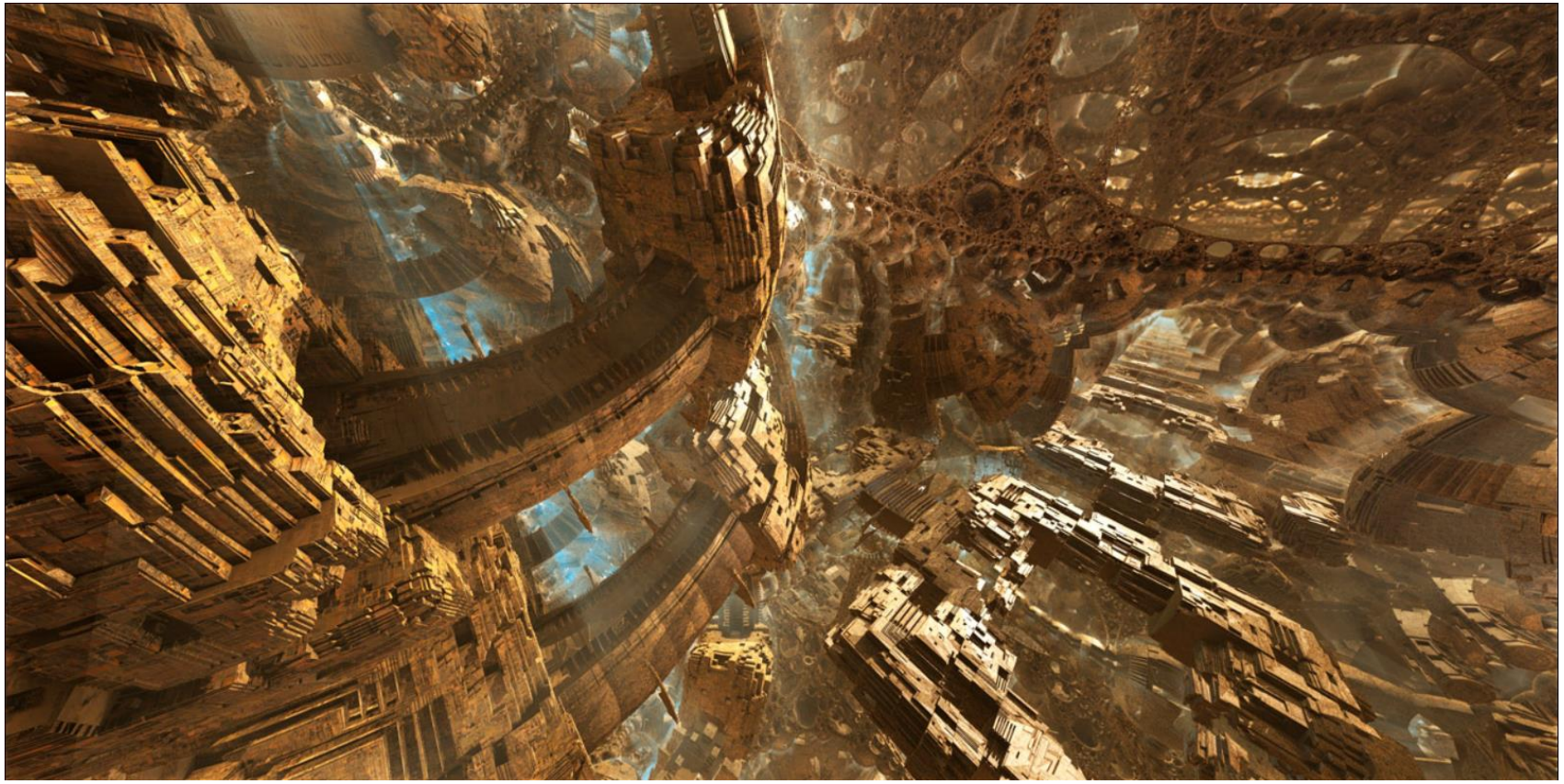
Один воксель соответствует ячейке на регулярной трехмерной сетке

Применяется для медицинской визуализации (КТ, МРТ), моделирования непрерывных сред и полей значений, расчета ненаправленного освещения

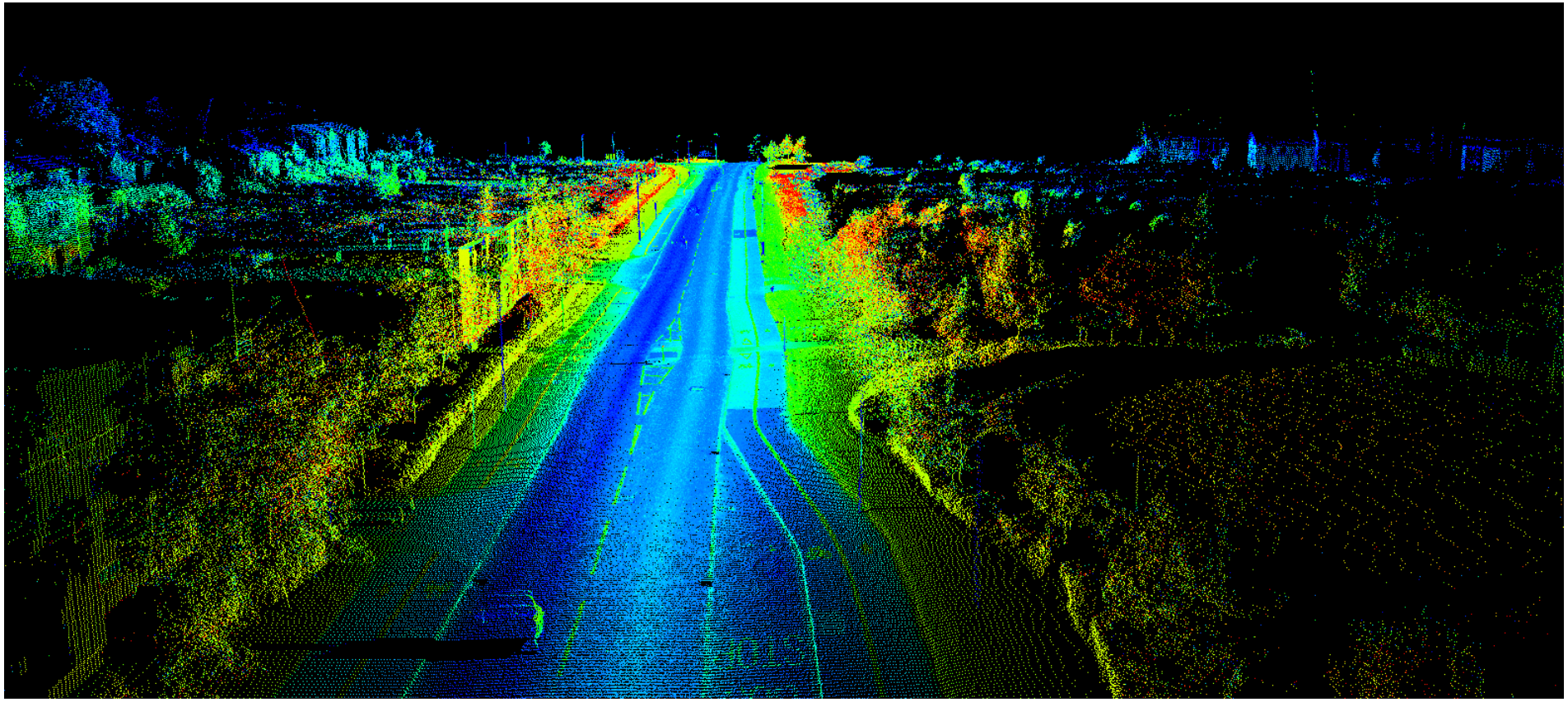
Фрактальная модель

Результат трассировки трехмерного фрактала

Размер и детализация не ограничены



Точечная модель



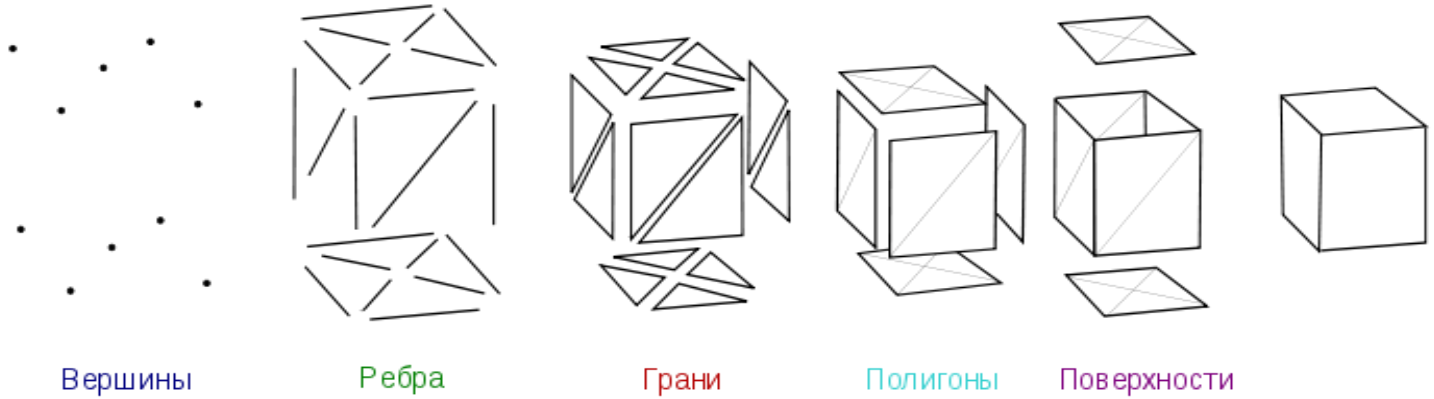
Поверхностная модель

Может быть представлена полигональной моделью

Широко применяется при визуализации трехмерных объектов

Полигональное представление модели

- Вершина
  - Точка в трехмерном пространстве
- Ребро
  - Объединяет две вершины
- Грань
  - Замкнутое множество ребер (как правило, треугольник)
- Полигон
  - Чаще всего равнозначен грани



Вершина

Содержит информацию о местоположении и атрибутах элемента поверхности

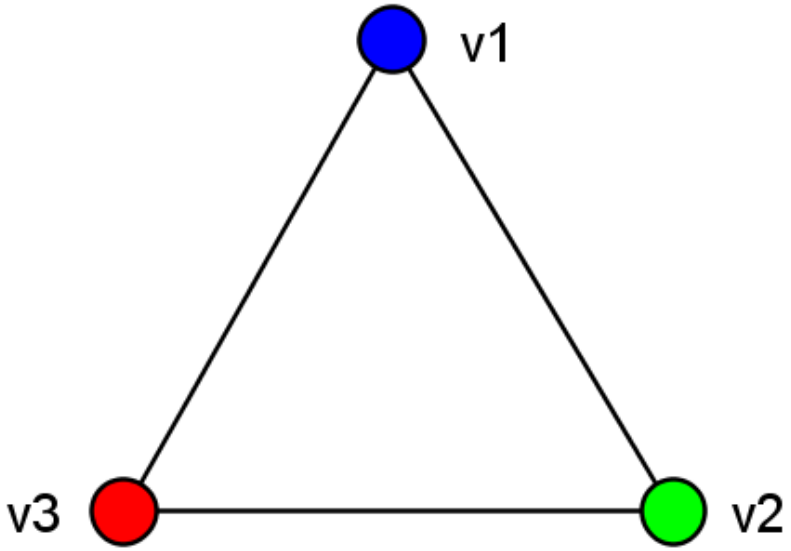
Атрибуты вершины необходимы при визуализации и расчетах.

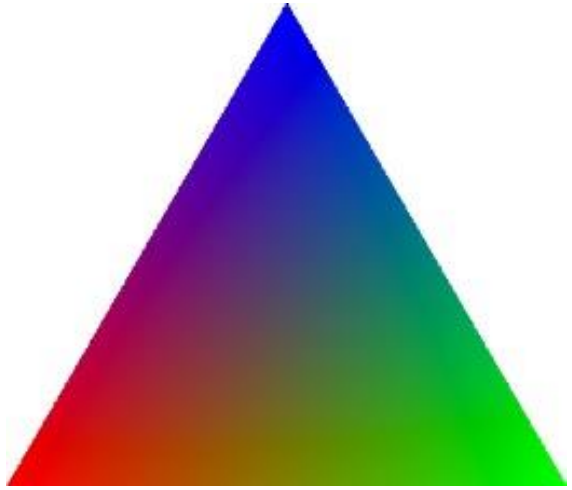
Набор вершин не может представлять ни поверхность, ни линию (без какой либо дополнительной информации)

Помимо позиции, может содержать атрибуты:

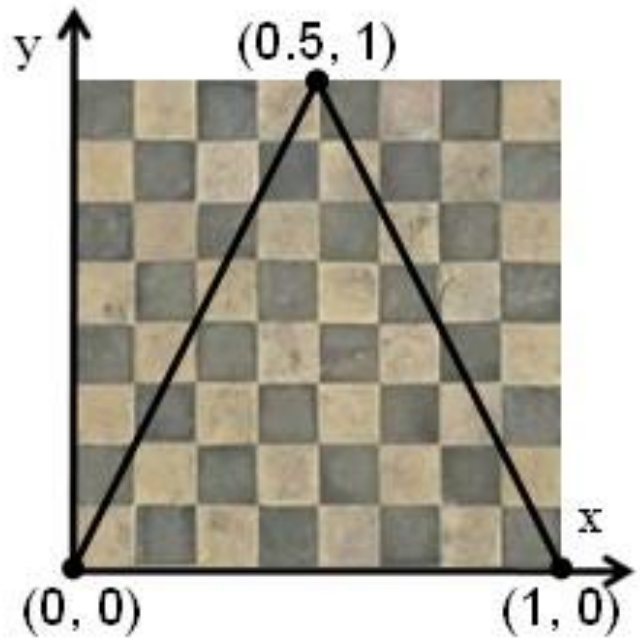
- Цвет (rgb)
- Нормаль (xy или xyz)
- Текстурные координаты (uv)
- …

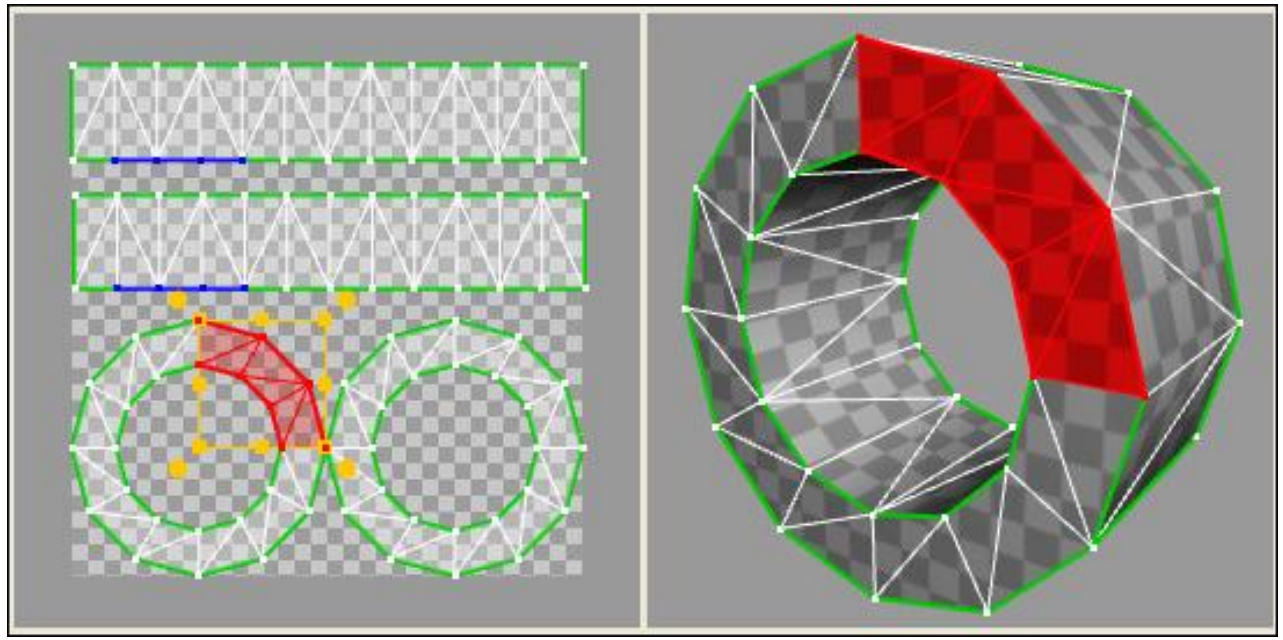
Атрибуты вершины: цвет



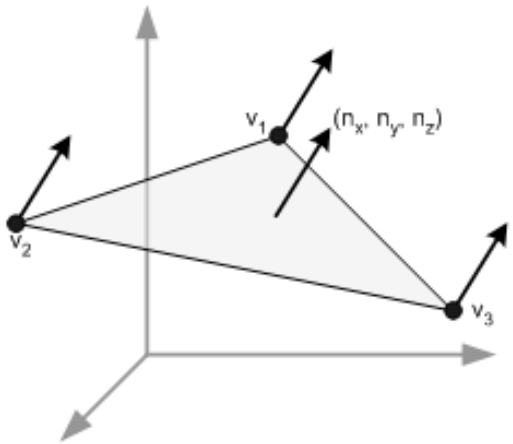


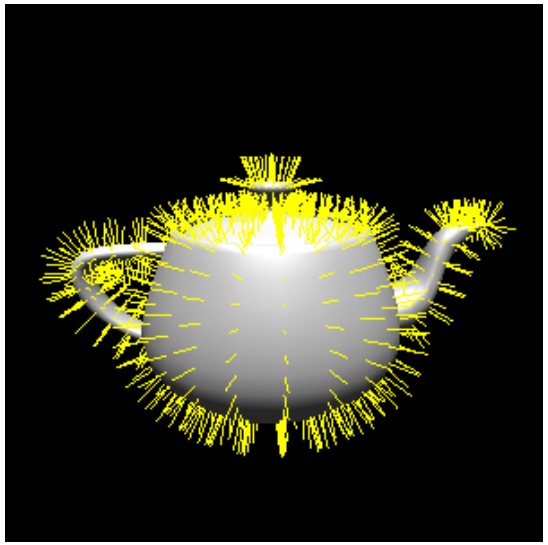
Атрибуты вершины: текстурные координаты

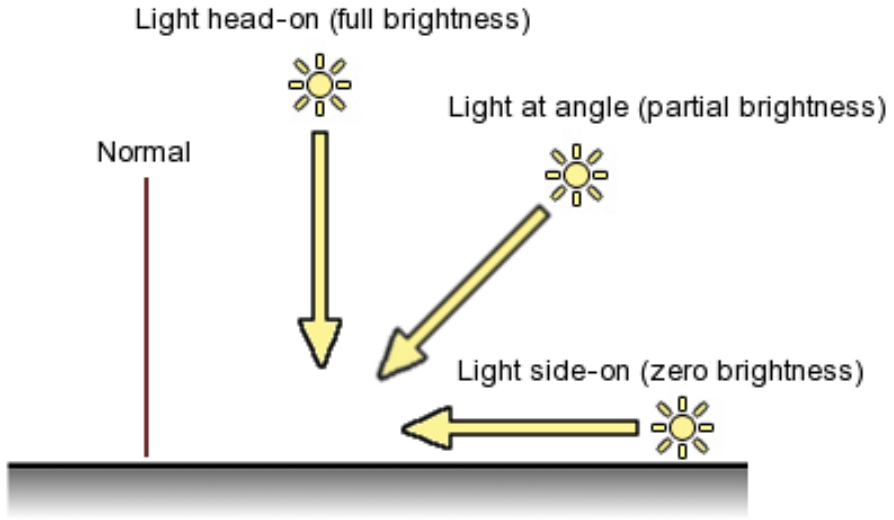




Атрибуты вершины: нормаль к поверхности







Топология полигональной модели

При визуализации полигональная сетка может быть представлена набором линий или граней.

Гранями обычно являются треугольники, т.к. другие виды полигонов не поддерживаются современными видеокартами.

Для отображения необходимо иметь информацию о связности вершин модели.

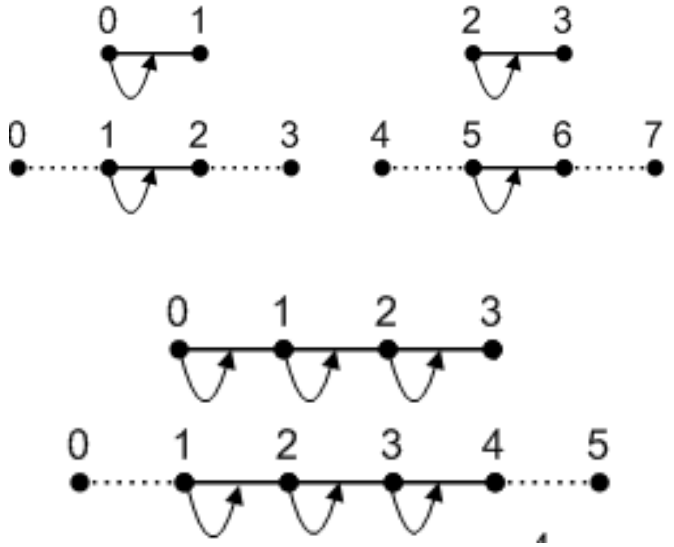
Построение линий

Line List

- Каждые две вершины могут задавать отрезок (вершины 1 и 2– один отрезок, вершины 3 и 4 – другой отрезок)

Line Strip

- Каждая последующая вершина может задавать отрезок (вершины 1 и 2 – первый отрезок, вершины 2 и 3 – второй отрезок)



Построение треугольников

Triangle List

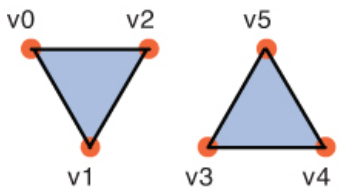
- Каждые три вершины могут задавать треугольник (грань) (v[n], v[n+1], v[n+2])

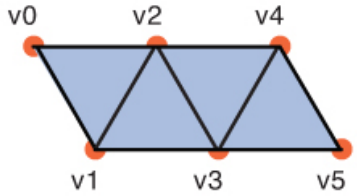
Triangle Strip

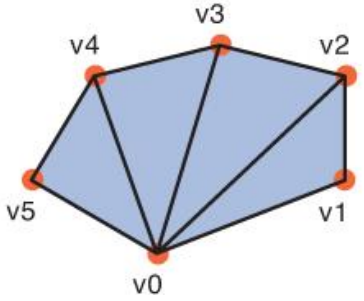
- Каждая последующая вершина вместе с двумя предыдущими вершинами задает новый треугольник (v[n], v[n-1], v[n-2])

Triangle Fan

- Каждая последующая вершина вместе с предыдущей и первой задает треугольник (v[n], v[n-1], v[0])



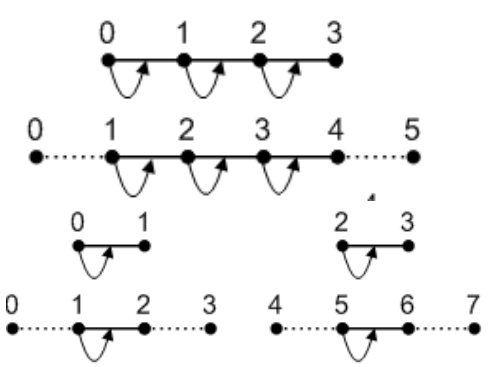


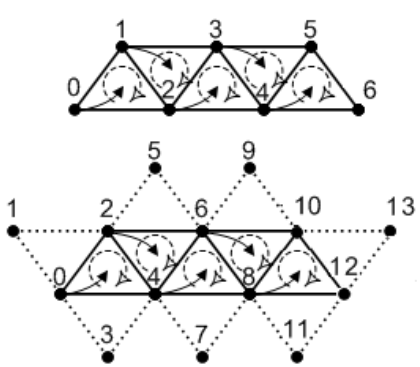


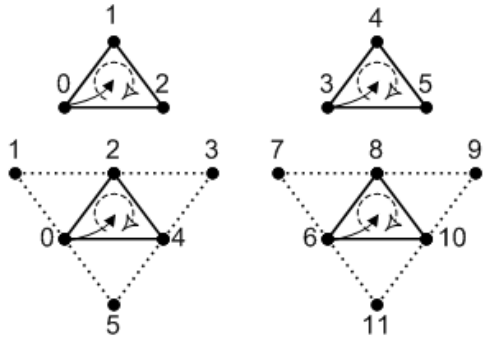
Использование индексов

Описанные выше методы применимы в ситуации, когда вершины расположены в определенном порядке (который варьируется в зависимости от типа используемого примитива)

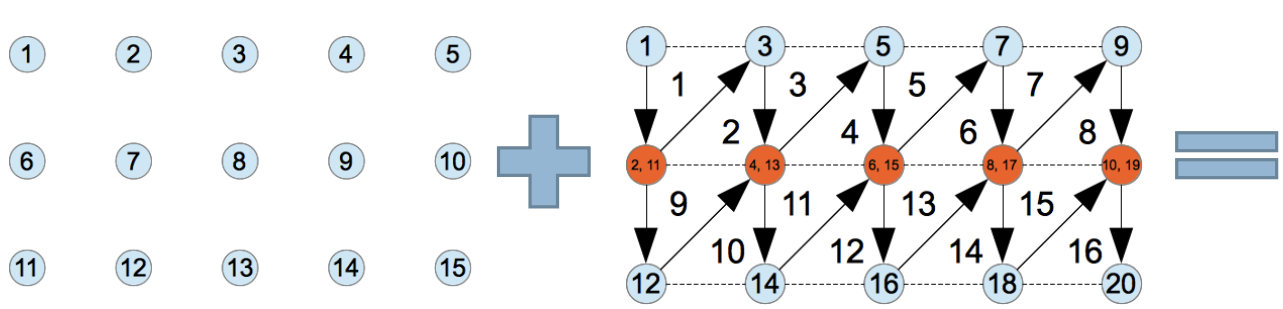
Использование индексов позволяет определять порядок обхода вершин вне зависимости от их местоположения



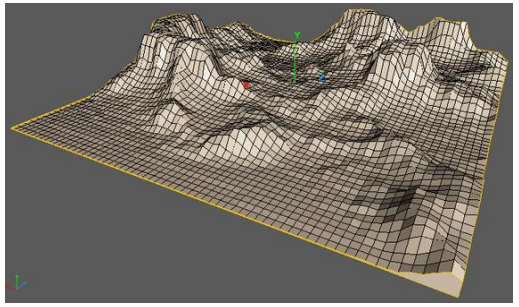




Пример



Вершины Индексы



Результат

Хранение полигональной модели

Формат OBJ (.obj)

- Наиболее распространенный формат хранения полигональной модели
- Простая структура данных
- Позволяет хранить:
  - Позицию, нормаль и текстурные координаты вершины
  - Линии и грани
  - Материал
- Недостатки:
  - Высокий размер файла
  - Ограниченный набор возможностей
- Спецификация:
  - https://www.cs.utah.edu/~boulos/cs3505/obj_spec.pdf

Формат Obj

```
# Список вершин, с координатами (x,y,z[,w]), w является не обязательным и по умолчанию 1.0.
v 0.123 0.234 0.345 1.0
v ...
...

# Текстурные координаты (u,v[,w]), w является не обязательным и по умолчанию 0.
# Текстурная координата по y может быть указана как 1 - v, и при этом по x = u
vt 0.500 -1.352 [0.234]
vt ...
...

# Нормали (x,y,z); нормали могут быть не нормированными. .
vn 0.707 0.000 0.707
vn ...
...

# Параметры вершин в пространстве (u [,v] [,w]); свободная форма геометрического состояния (смотри ниже)
vp 0.310000 3.210000 2.100000
vp ...
...

# Определения поверхности (сторон) (смотри ниже)
f 1 2 3 f 3/1 4/2 5/3 f 6/4/1 3/5/3 7/6/5 f 6//1 3//3 7//5
f ...
...

# Группа
g Group1 ...

# Объект
o Object1
```

Пространственные преобразования

Каждая вершина модели находится в собственном пространстве, называемым «**пространством модели**»

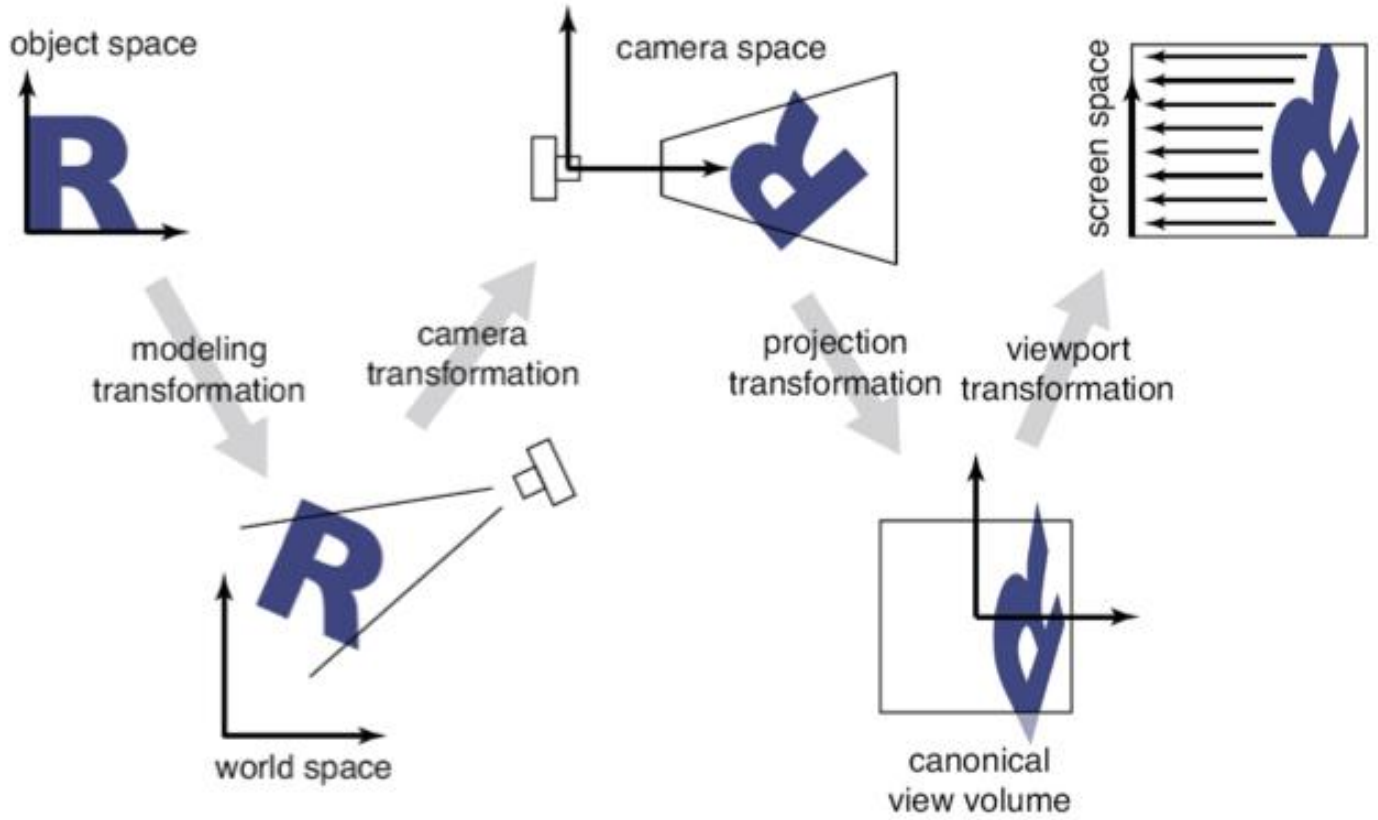
Пространственные преобразования применяются если:

- Необходимо задать некое пространственное отношение между моделями
- При проецировании модели в пространство экрана

Для определения пространственного отношения между моделями необходимо переместить их в общее пространство

- Такое пространство называют «**пространством мира**» (World Space)

Пространственные преобразования



Системы координат

Пространственное отношение между моделями может быть задано **смещением**, **вращением** или **масштабированием**.

- Такие операции называются **преобразованием**.
- Следует рассматривать преобразование как переход от одного пространства к другому

Когда все объекты преобразованы в единое пространство (мировое), их вершины будут заданы относительно координатной системы мира

Системы координат

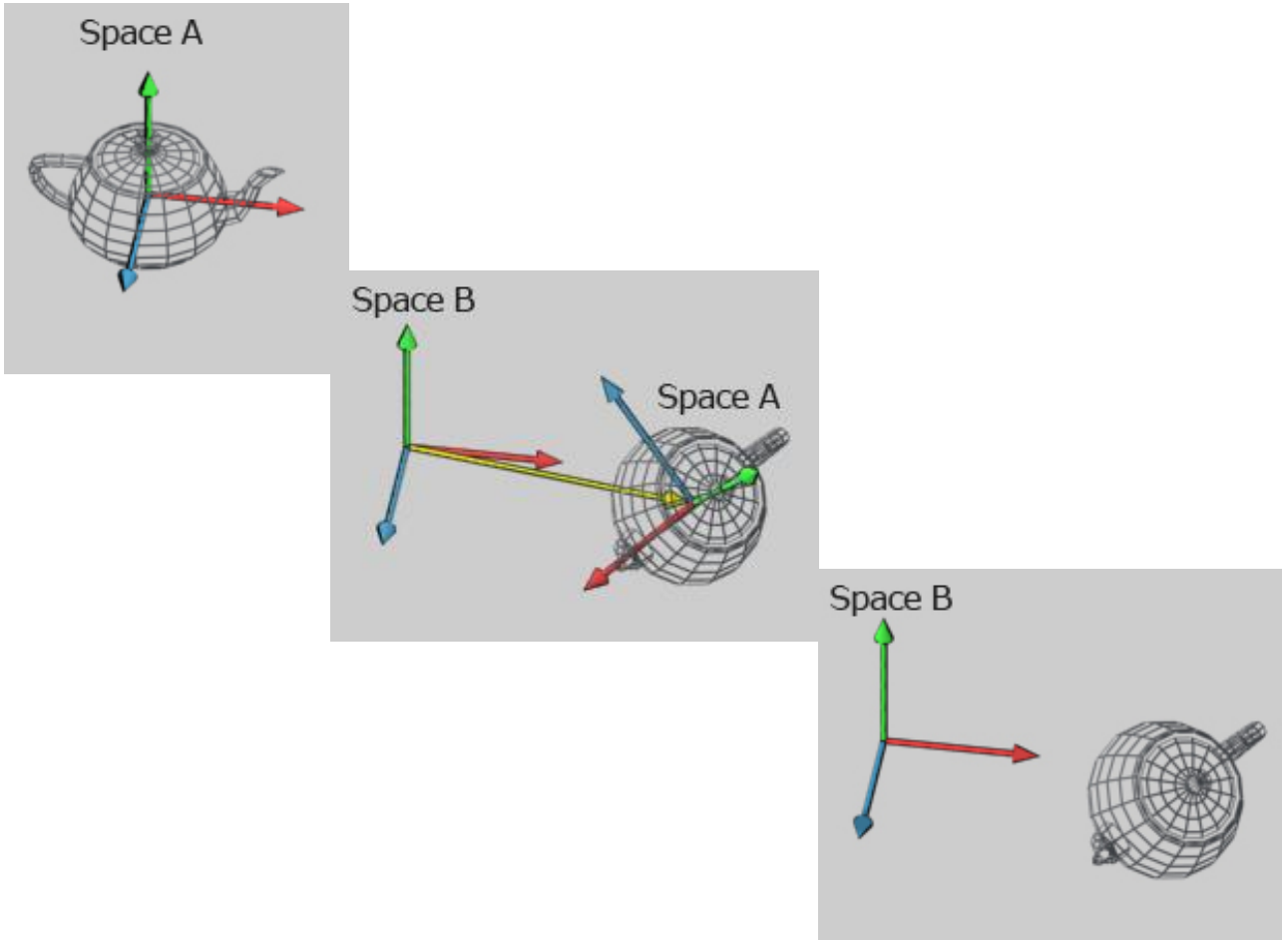
Допустим, имеется пространство A, в котором задана модель

Мы хотим применить трансформацию, которая переместит все из пространства A на новую позицию (пространство B)

Перемещение модели происходит при помощи применения трансформации к каждой вершине модели

До преобразования все вершины задавались относительно пространства A, после преобразования – относительно пространства B

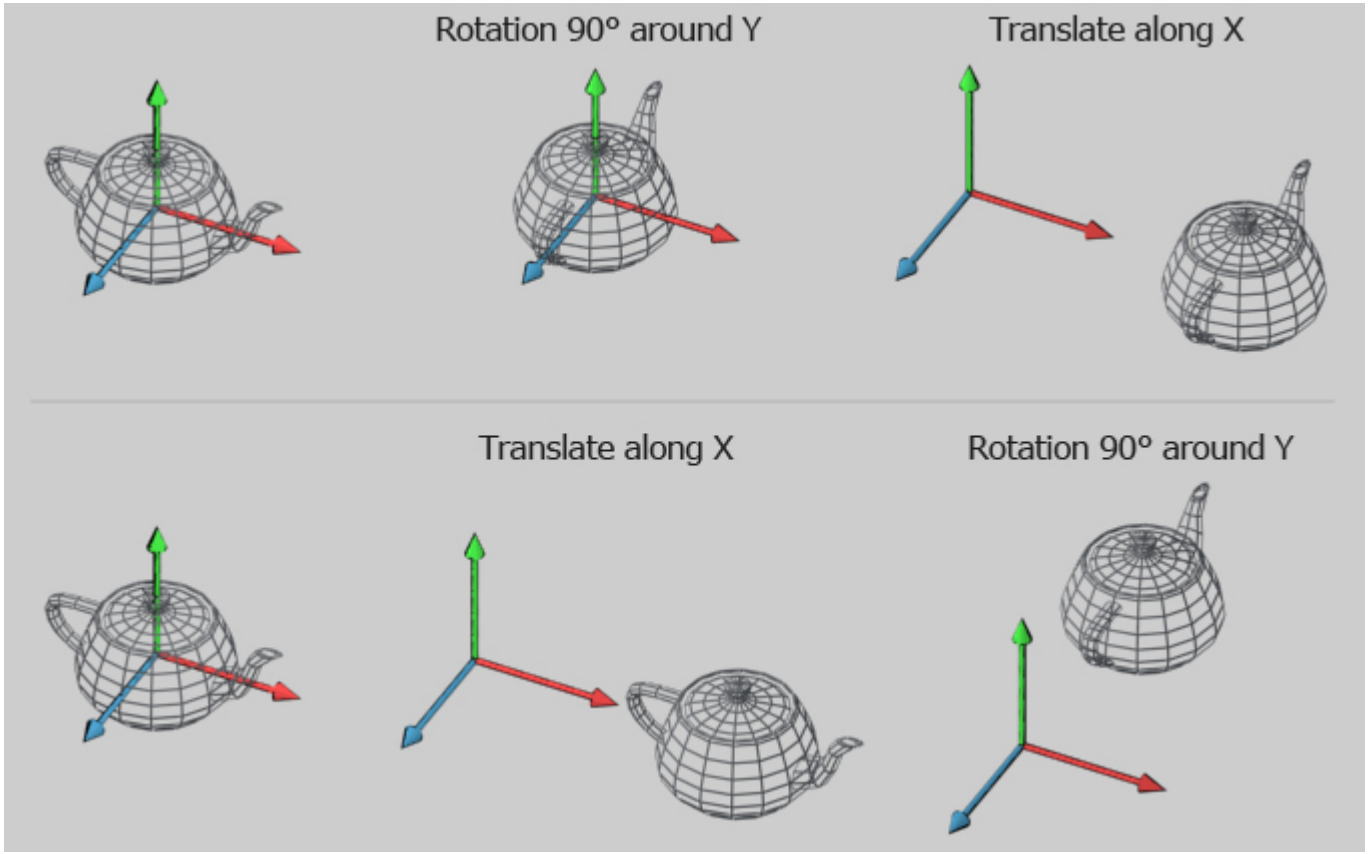
После преобразования пространство A оказалось потеряно в пространстве B, но его можно восстановить, инвертировав преобразование



Последовательные трансформации

Каждая трансформация относительна к исходной

- Поэтому, порядок трансформаций очень важен



Однородные координаты

В полигональной модели координаты вершины обычно представлены в виде декартовых координат: *(x, y, z)*

Для применения матрицы трансформации необходимо перевести координаты вершины из декартовых в однородные

Декартовы координаты: $[x \quad y \quad z]$

Однородные координаты: $[wx \quad wy \quad wz \quad w]$

- Где *w* – действительное число, не равное 0

Алгоритмы OpenGL по отсечению и растеризации работают в декартовых координатах, но перед этим все преобразования производятся в однородных координатах. Переход от однородных координат в декартовы координаты осуществляется аппаратно.

Однородные координаты

$[x \quad y \quad z \quad 1]$ - точка, где (x, y, z) – декартовы координаты

$[x \quad y \quad z \quad 0]$ - точка, где (x, y, z) – радиус-вектор

Обратный перевод вершины из однородных координат в декартовы координаты осуществляется следующим образом:

- $[x_h \quad y_h \quad z_h \quad w_h]$ - однородные координаты
- $\left[\frac{x_h}{w_h} \quad \frac{y_h}{w_h} \quad \frac{z_h}{w_h}\right]$ - декартовы координаты

Матрицы трансформации

В компьютерной графике обычно используются матрицы трансформации 4x4

Для трансформации вершины в однородных координатах (x,y,z,w) достаточно умножить ее на матрицу трансформации

- Порядок умножения важен, и может различаться в разных библиотеках

Мата трансформации:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax + by + cz + dw \\ ex + fy + gz + hw \\ ix + jy + kz + lw \\ mx + ny + oz + pw \end{bmatrix}$$

```
glm::mat4 myMatrix;
glm::vec4 myVector;
glm::vec4 transformedVector = myMatrix * myVector;
```

Матрица перемещения

Матрица перемещения, наверное, самая простая из всех:

$$\begin{bmatrix} 1 & 0 & 0 & Translation.x \\ 0 & 1 & 0 & Translation.y \\ 0 & 0 & 1 & Translation.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Перемещение вектора (10,10,10) на 10 единиц по оси X:

$$\begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 10+0+0+10 \\ 0+10+0+0 \\ 0+0+10+0 \\ 0+0+0+1 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

```
#include <glm/transform.hpp>
glm::mat4 myMatrix = glm::translate(10.0f, 0.0f, 0.0f);
glm::vec4 myVector(10.0f, 10.0f, 10.0f, 0.0f);
glm::vec4 transformedVector = myMatrix * myVector;
```

Матрица масштабирования

Предназначена для умножения компонент вектора на определенное значение:

$$\begin{bmatrix} Scale.x & 0 & 0 & 0 \\ 0 & Scale.y & 0 & 0 \\ 0 & 0 & Scale.z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Умножение компонент вектора (10,10,10) на 1,2 и 3 по осям x,y и z:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 * 1 + 0 + 0 + 0 \\ 0 + 10 * 2 + 0 + 0 \\ 0 + 0 + 10 * 3 + 0 \\ 0 + 0 + 0 + 1 * 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 20 \\ 30 \\ 1 \end{bmatrix}$$

glm::mat4 myScalingMatrix = glm::scale(2.0f, 2.0f ,2.0f);

Матрица поворота

Матрица вращения используется для поворота множества точек внутри системы координат. В то время как отдельным точкам назначаются новые координаты, их относительные расстояния не изменяются.

Все вращения определяются с помощью тригонометрического «синуса» и «косинуса»

При использовании углов эйлера выполняется последовательность поворотов по осям X,Y и Z

Матрица вращения по оси X:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Матрица вращения по оси Y:

$$\begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Матрица вращения по оси Z:

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
glm::vec3 myRotationAxis( ??, ??, ??);
glm::rotate( angle_in_degrees, myRotationAxis );
```

Последовательность трансформаций

Для пения трансформации (масштаб+поворот+смещение) необходимо последовательно перемножить соответствующие матрицы, и помножить результат на вершину.

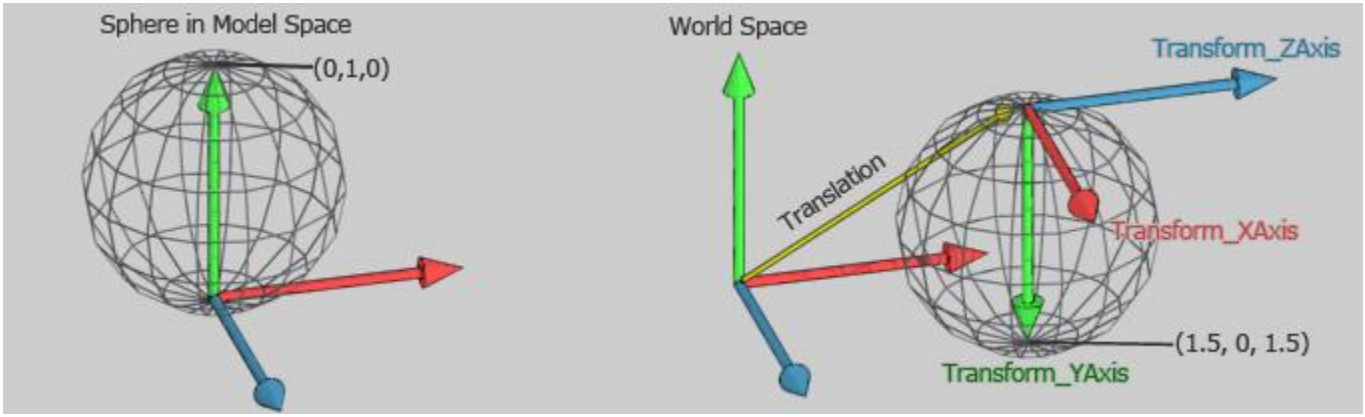
Порядок умножения очень важен.

Масштабирование, затем поворот, затем смещение:

- *TransformedVector = TranslationMatrix * RotationMatrix * ScaleMatrix * OriginalVector;*

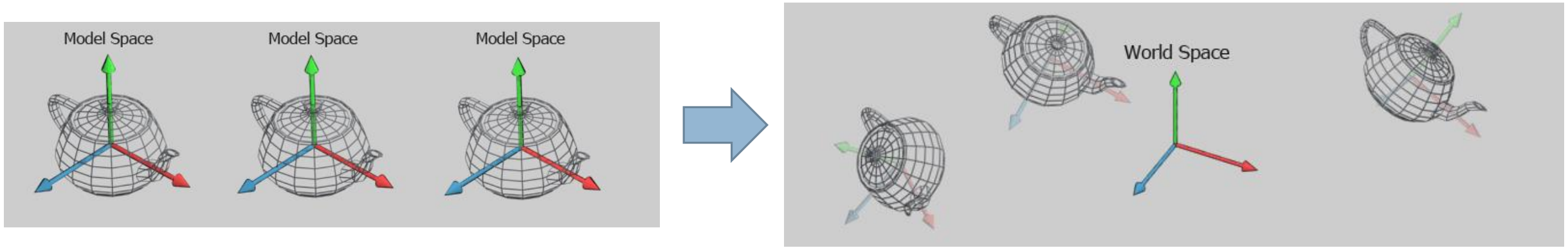
Масштабирование, затем смещение, затем поворот уже смещенного объекта вокруг начала координат:

- *TransformedVector = RotationMatrix * TranslationMatrix * ScaleMatrix * OriginalVector;*



Пространства модели, мира и вида

Все объекты имеют собственную позицию и ориентацию, поэтому у каждого из них есть своя матрица преобразования в пространство мира



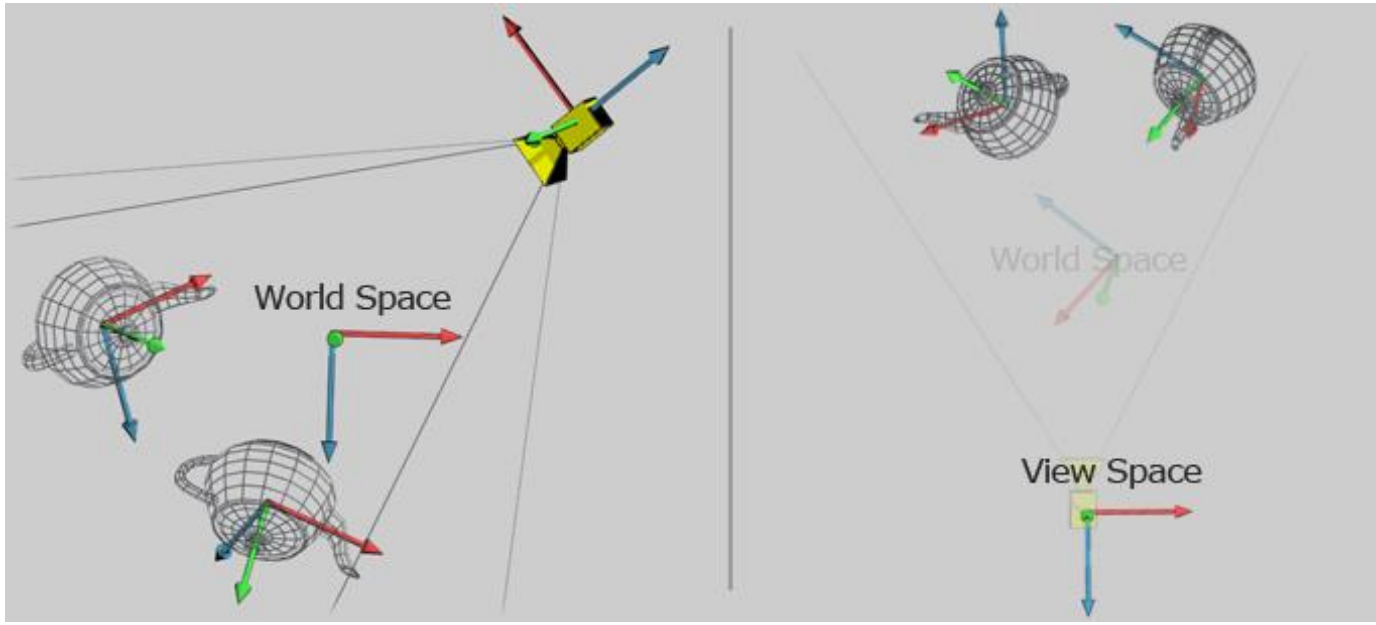
Пространство вида (камеры)

Теперь, когда все объекты на своих местах, необходимо спроецировать их на экран.

- Сначала все объекты переносятся в **пространство вида** (View Space)

В пространстве вида все координаты объектов задаются относительно камеры

Для выполнения преобразования координата вершины должна быть домножена на матрицу вида:




```
glm::mat4 CameraMatrix = glm::LookAt(cameraPosition, cameraTarget, upVector);
```

Пространство экрана

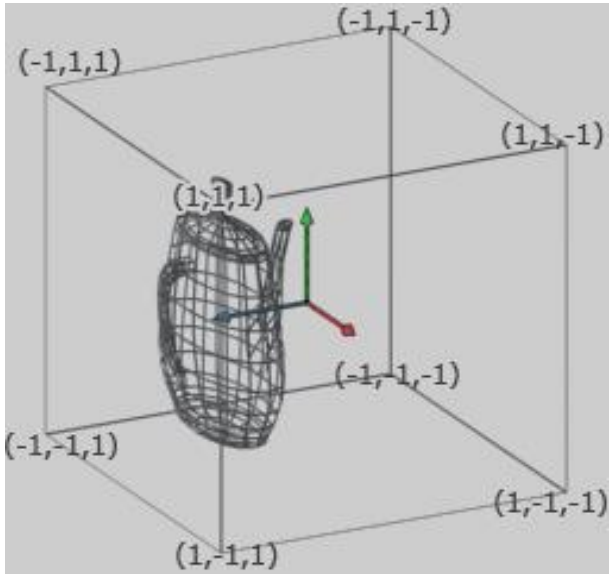
Сцена теперь в наиболее удобном для проецирования пространстве – **пространстве вида**

Перед получением плоского изображения необходимо произвести финальное преобразование в **пространство экрана** (Screen Space)

Пространство экрана имеет форму куба, размеры которого находятся между 1 и -1 по каждой из осей

Это пространство очень удобно:

- Для отсечения – все вершины вне диапазона 1:-1 находятся вне зоны видимости камеры
- Для преобразования в изображение – достаточно отбросить z координату



Пространство экрана

Для того, чтобы преобразовать пространство вида в пространство экрана нужна другая матрица, значения которой зависят от типа выбранной проекции

Обычно используется два вида проекции:

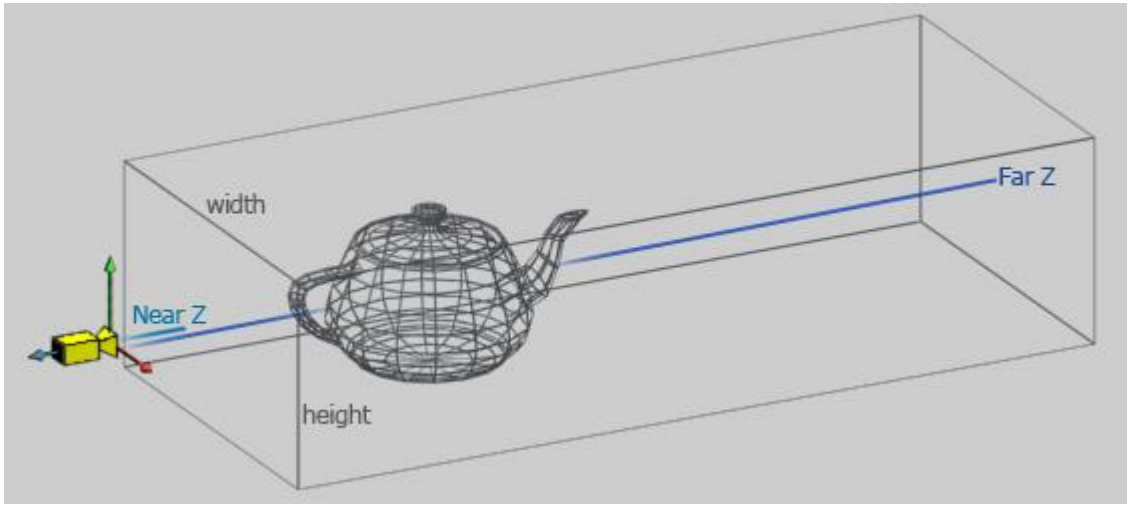
- Ортографическая проекция
- Перспективная проекция

Ортографическая проекция

Ортографическая проекция - изображение какого либо предмета на плоскости так, как бы он представлялся наблюдателю если бы он смотрел на него с бесконечного расстояния.

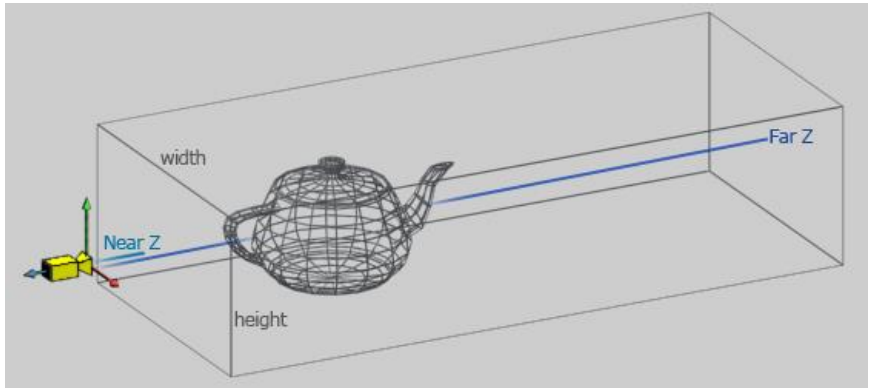
Для того чтобы задать такую проекцию необходимо указать размеры области, которую камера может видеть. Обычно для этого требуется:

- Указать ширину и высоту по осям X и Y
- Указать ближнюю и дальнюю плоскости отсечения по оси Z

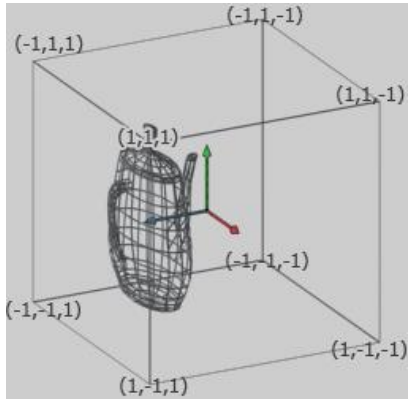


Ортографическая проекция: построение матрицы

Используя полученные значения, мы можем создать матрицу трансформации которая сможет преобразовать прямоугольный параллелепипед в куб:



$$\begin{bmatrix} \frac{1}{width} & 0 & 0 & 0 \\ 0 & \frac{1}{height} & 0 & 0 \\ 0 & 0 & -\frac{2}{Z_{far}-Z_{near}} & -\frac{Z_{far}+Z_{near}}{Z_{far}-Z_{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



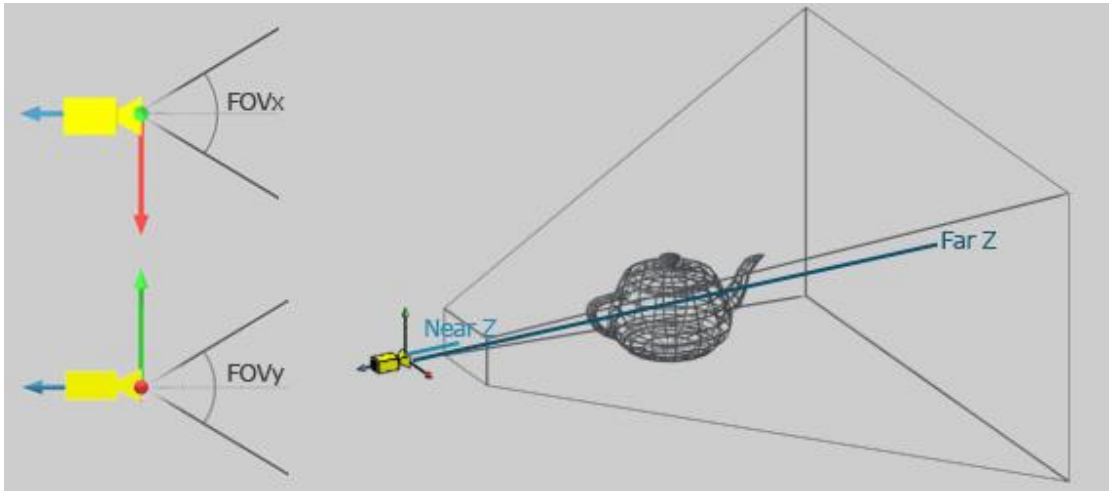
Перспективная проекция

Идея аналогична ортографической проекции, но теперь зона видимости – пирамида, преобразование из которой чуть более сложно

Умножения матриц в этом случае недостаточно, и после преобразования надо поделить в полученном векторе xyzw компоненты xyz на w (так как w не равен 1)

Для того чтобы задать такую проекцию необходимо указать размеры пирамиды, которую камера может видеть. Обычно для этого требуется:

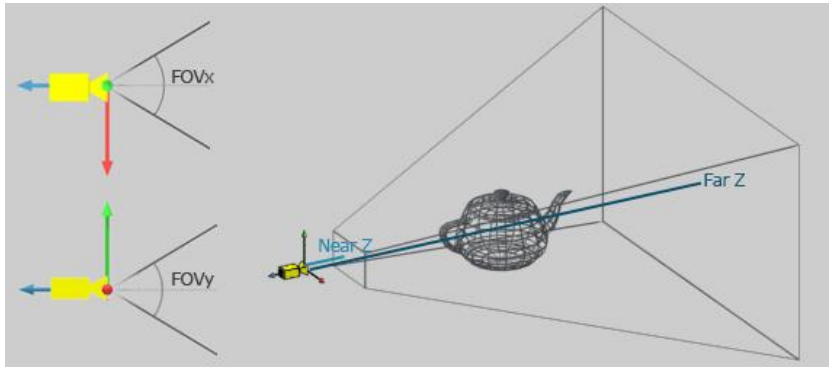
- Указать горизонтальный и вертикальный углы видимости
- Указать ближнюю и дальнюю плоскости отсечения по оси Z



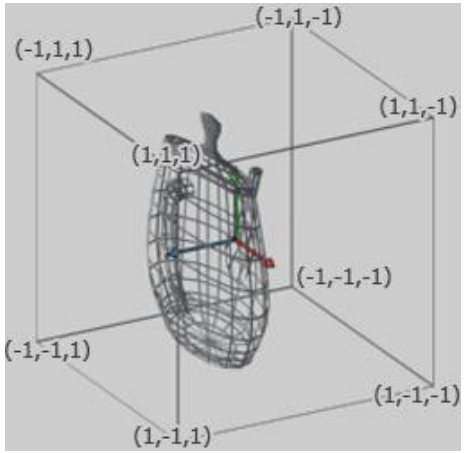

```
glm::mat4 projectionMatrix = glm::perspective(
    FoV,         // Горизонтальное Поле Вида в градусах.
    4.0f / 3.0f, // Соотношение сторон. Зависит от размера вашего окна. Например, 4/3 == 800/600 == 1280/960
    0.1f,        // Ближнее поле отсечения. Его нужно задавать как можно большим, иначе будут проблемы с точностью.
    100.0f       // Дальнее поле отсечения. Нужно держать как можно меньшим.
);
```

Перспективная проекция: построение матрицы

Используя полученные значения, мы можем создать матрицу трансформации которая сможет преобразовать пирамиду в куб:

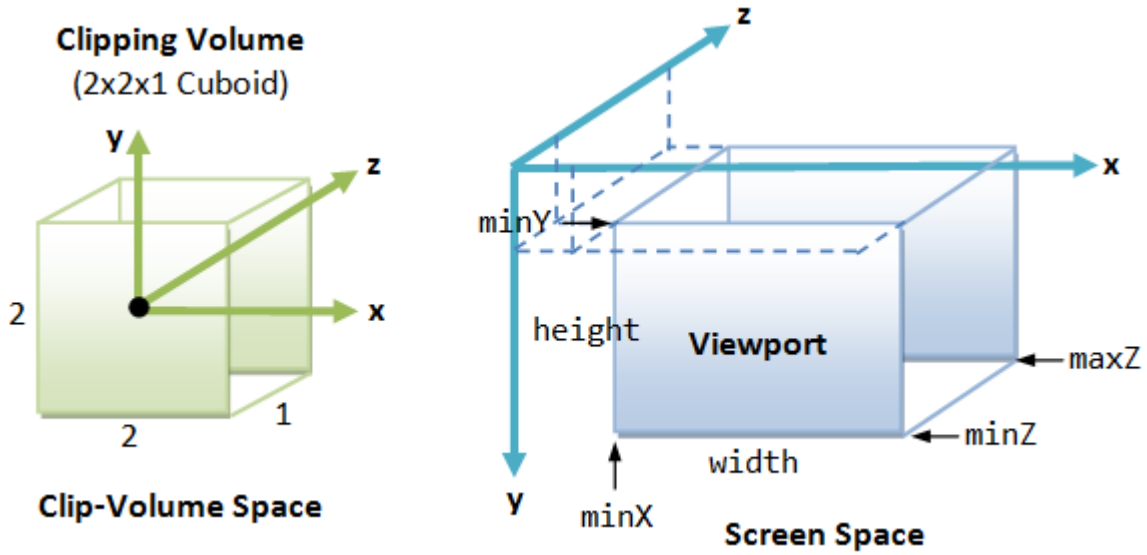


$$\begin{bmatrix} \tan^{-1}(\frac{FOVx}{2}) & 0 & 0 & 0 \\ 0 & \tan^{-1}(\frac{FOVy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{Z_{far}+Z_{near}}{Z_{far}-Z_{near}} & -\frac{2\,(Z_{near}\,Z_{far})}{Z_{far}-Z_{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



Viewport Space

Для отображения на экране необходимо сделать последний шаг – преобразовать из координат -1:1 в экранные координаты



Спасибо за внимание!