# 1. Обзор фреймворков

## 1.1. ORK (OpenGL rendering kernel)

Небольшой фреймворк для работы с OpenGL. Реализованы следующие возможности:

- автоматическая сборка мусора, основанная на умных указателях;
- динамическая загрузка объектов по xml-описанию;
- создание графа задач и его выполнение.

Также содержит примитивы линейной алгебры и примитивы для работы с OpenGL.

## 1.2. Proland

Фреймворк для эффективного и корректного отображения местности на глобусе, включая рельеф, дороги и водоемы. В своей реализации активно опирается на ORK.

Состоит из ядра и набора плагинов. Каждый плагин содержит объекты для работы с некоторой предметной областью:

- terrain для рельефа;
- atmo для атмосферы;
- graph для графов (и дорог в частности)
- и т.д.

Большинство плагинов не зависят друг от друга.

Для решения поставленной задачи (определение высоты точки по ее угловым координатам) необходимо использовать плагин terrain и его зависимости.

# 2. Процесс сборки

Для корректной работы Proland необходимо подключить правильно сконфигурированную библиотеку LIBTIFF. При сборке LIBTIFF из исходных файлов требуется в конфигурационном файле nmake.opt раскомментировать следующие строки, указав корректный путь к библиотеке zlib:

```
ZIP_SUPPORT      = 1
ZLIBDIR          = <some path>
ZLIB_INCLUDE = -I$(ZLIBDIR)
ZLIB_LIB         = $(ZLIBDIR)/zlib.lib
```

Фреймворк ORK имеет собственную реализацию `shared_ptr`. Для того, чтобы использовать стандартный указатель, следует выставить директиву препроцессора USE_SHARED_PTR. Также этот фреймворк включает в себя небольшую библиотеку для чтения xml – TinyXML. Она может использовать свою реализацию строк. Для того, чтобы использовать стандартные строки, следует выставить директиву препроцессора TIXML_USE_STL.

Итоговые директивы препроцессора:

- USE_SHARED_PTR;
- _CRT_SECURE_NO_WARNINGS (только для proland);
- PROLAND_API=__declspec(dllexport/dllimport);
- ORK_API=__declspec(dllexport/dllimport);
- TIXML_USE_STL.

# 3. Основные понятия

Terrain плагин позволяет работать с растровыми и векторными данными на плоскости и на сфере. Для работы со сферой применяется методика *cube mapping*. Высотные данные должны находиться в файле DEM.dat для модели на плоскости или DEM[1-6].dat для сферической модели. Каждый файл соответствует некоторой грани куба (для сферического случая).

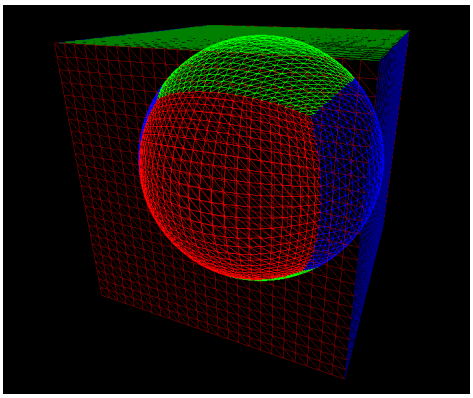

Рис. 1: Соответствие граней куба областям сферы

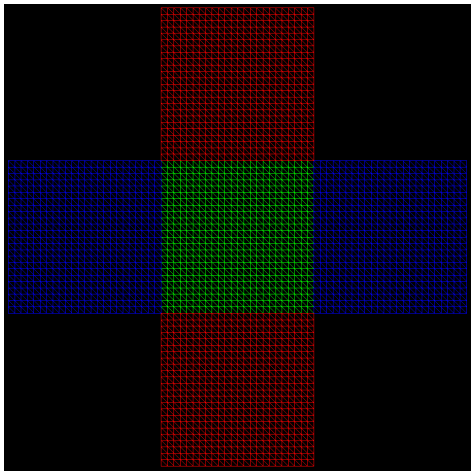

Рис. 2: Развертка куба

Данные для некоторой квадратной области (далее: *исследуемая область*) хранятся с помощью *квадродерева*. Каждый уровень состоит из одного или более *quad*. *Тайлом* называется информация, соответсвующая некоторому quad. Первый уровень квадродерева содержит ровно один quad. При переходе на следующий уровень quad либо разбивается на 4 дочерних, либо не разбивается вообще.

## 3.1. Системы координат

При работе с quad и тайлами используются две системы координат: логическая и физическая. В каждой из них quad задается тройкой $(l, x, y)$, где $l$ – номер уровня в квадродереве, $x, y$ – координаты левого нижнего угла quad в текущей системе координат.

**Логическая система координат** Началом координат является левый нижний угол. Правый верхний угол имеет координаты $(2^l, 2^l)$, где $l$ – порядковый номер уровня в квадродереве. Каждый quad имеет размеры 1x1, таким образом, число quad на уровне – $4^l$.

**Физическая система координат** Началом координат является центр квадрата. Левый нижний угол имеет координаты $(-\frac{l}{2}, -\frac{l}{2})$, правый верхний соответственно $(\frac{l}{2}, \frac{l}{2})$, где $l$ – длина стороны исследуемой области.

**Географическая система координат** Полярная система координат, используемая для позиционирования на земле. Широта лежит в диапазоне от $-90$ градусов (или 90 градусов южной широты) до 90 градусов (или 90 градусов северной широты). Долгота лежит в диапазоне от $-180$ градусов (или 180 градусов западной долготы) до 180 градусов (или 180 градусов восточной долготы).

## 3.2. Хранилище тайлов (`TileStorage`) и кеши (`TileCache`)

С каждым quad можно ассоциировать некоторую растровую или векторную информацию. Растровая информация может храниться в двух формах: в виде 2D-текстур для дальнейшего эффективного отображения (`GPUTileStorage`) или в виде массивов чисел (`CPUTileStorage`). Хранилище имеет заранее заданный размер. Тайлы могут иметь непустую границу (в представленных DEM-файлах она составляет 2 пикселя). Это порождает некоторую избыточность, но бывает полезно в некоторых случаях (например, чтобы избежать артефактов при фильтрации текстур или генерации соседних тайлов).

С каждым хранилищем можно ассоциировать кеш, который будет следить за тем, какие из тайлов используются. Если тайл не используется, он может быть удален кешом из хранилища в произвольный момент времени. Тайл можно получить методом `TileCache::getTile`, после чего число владельцев тайла увеличивается на один. После вызова метода `TileCache::putTile` число владельцев уменьшается на один. Тайл может быть корректно удален (например, при завершении программы и очистке памяти), только если у него нет ни одного владельца.

## 3.3. Производители тайлов (`TileProducer`)

Для хранения высотных данных используется дельта-кодирование. Таким образом, тайл высотных данных (*высотный тайл*, *elevation tile*) уровня $l$ может быть построен из высотного тайла уровня $l-1$ и соответствующего *добавочного тайла* (*residual tile*) уровня $l$. Для этого родительский высотный тайл разбивается на 4 дочерних и к ним добавляются добавочные тайлы.

Для построения высотных тайлов существует два класса: `ElevationProducer` и `CPUElevationProducer`. Первый используется для

создания 2D-текстур и хранит тайлы в `GPUTileStorage`. Второй используется для хранения высот в виде отдельных значений и хранит тайлы в `CPUTileStorage`. DEM-файлы содержат в себе заранее подготовленные добавочные тайлы, которые могут быть считаны при помощи класса `ResidualProducer`.

Один и тот же кеш может быть использован несколькими производителями. Для этого каждому производителю дается уникальный номер, а тайл внутри кеша характеризуется четверкой чисел $(p, l, x, y)$, где $p$ – номер производителя, $l$ – номер уровня в квадродереве, $x, y$ – координаты левого нижнего угла тайла в текущей системе координат.

## 3.4. Задачи (`Task`) и графы задач (`TaskGraph`)

Так как выполение некоторых действий может занимать достаточно продолжительное время (например, чтение добавочных тайлов из файла с диска), активно применяются многопоточность и отложенные вычисления. Для этого создаются задачи, которые будут выполнены планировщиком в некоторый момент времени. Класс `MultithreadScheduler` представляет собой реализацию планировщика для нескольких потоков.

Некоторые действия должны происходить в некотором порядке (например, для построения высотного тайла, нужно построить высотный тайл предыдущего уровня, а также один или несколько добавочных тайлов). Для решения этой проблемы используются графы задач. Между задачами можно создавать зависимости, таким образом давая планировщику возможность понять, можно ли выполнить данную задачу или нет.

Например, для построения высотного тайла уровня $l$, нужно построить родительский тайл (если он еще не построен), а также один или несколько добавочных тайлов. Создается граф задач, в который добавляются задачи построить высотный тайл, задачи построить добавочные тайлы, добавляются соответсвующие зависимости. В тот момент, как запрашивается родительский высотный тайл, в граф будет добавлена задача его создания и зависимость от результата его исполнения. Если для построения родительского тайла требуется построить еще какие-то тайлы, задачи на их построение также будут добавлены в граф.

Одно и то же задание можно добавлять в несколько графов задач. Это может потребоваться, например, если некоторый тайл уже кем-то

запрошен, но также требуется для выполнения некоторого действия.

## 3.5. DEM-файлы

DEM-файл хранит в себе заранее подготовленные добавочные тайлы и дополнительную служебную информацию. В начале файла идет заголовок, в нем записаны следующие значения:

1. минимальный уровень в квадродереве;
2. максимальный уровень в квадродереве;
3. размер тайлов в пикселях минус 5 [1];
4. начальный уровень в квадродереве;
5. и другое.

После заголовка идет массив сдвигов, по которым располагаются добавочные тайлы. Каждый тайл записан в формате TIFF. Сдвиг для конкретного тайла может быть вычислен, как функция от идентификатора тайла $(l, x, y)$, а также значений, указанных в заголовке.

Некоторая область может описывать не одним DEM-файлом. Например, если есть уточняющие данные для некоторого региона, то соответсутвующее им квадродерево может быть включено в исходное.

**Создание собственных DEM-файлов** Для самостоятельного создания DEM-файлов используются методы `preprocessDem` для случая плоскости и `preprocessSphericalDem` для случая сферы. В первом случае генерируется ровно один DEM.dat файл, во втором генерируются DEM[1-6].dat файлы. На вход этому методу передается некоторая реализация интерфейса `InputMap`. Метод `InputMap::getValue` выдает значение пикселя по его координатам.

Таким образом, существует возможность подгатавливать DEM-файлы из своих данных. Например, это может быть даже текстовый файл, в котором записаны координаты точек и их высоты.

[1] Proland внутри себя активно использует тот факт, что граница тайла имеет ширину в два пикселя. Таким образом, минимально возможный тайл имеет размер 5x5.

# 4. Методика решения

## 4.1. Высотные запросы

Метод `CPUElevationProducer::getHeight` позволяет отвечать на высотные запросы с некоторой точностью. Координаты точки задаются в физической системе координат.

Процесс перевода координат в полярной системе координат (широта-долгота) в локальную физическую систему координат выглядит следующим образом:

1. перевести координаты точки из полярной системы в декартову;
2. определить, на какой грани куба находится образ точки;
3. перевести декартовы координаты точки в систему координат грани куба (домножить на матрицу поворота);
4. построить образ точки на грани куба (методом `SphericalDeformation::deformedToLocal`).

## 4.2. Курсор

Класс `elevation_cursor` представляет из себя курсор, обладающий следующим интерфейсом:

- `void set_position(const lat_lon_d&)` – установить позицию курсора в географической системе координат;
- `double get_current_height() const` – узнать высоту с текущим уровнем детализации;
- `size_t get_current_level() const` – узнать текущий уровень детализации;
- `void leave_request(size_t)` – оставить запрос на повышение уровня детализации.

Последний метод принимает на вход целочисленные значения в диапазоне от минимального уровня квадродерева до максимального.