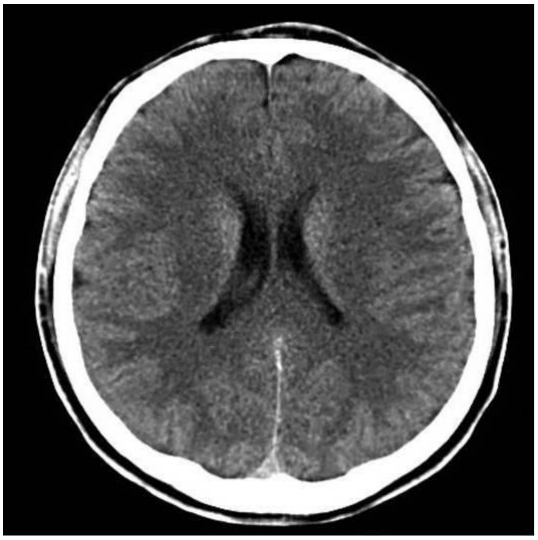
Лабораторная работа №2 "Визуализация томограмм"

Вторая лабораторная работа посвящена визуализации томограммы средствами OpenGL. Обратите внимание, что последующее руководство никак не может заменить справочник и лекции по OpenGL, при написании руководства подразумевается, что студенты прослушали лекции и имеют под рукой необходимую литературу.

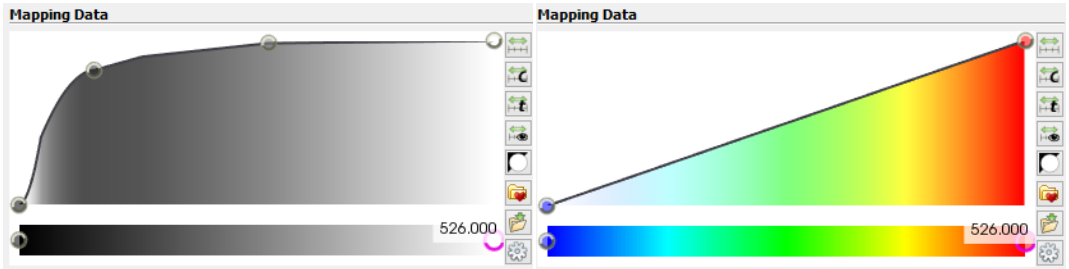
Томография – это получение послойного изображения внутренней структуры объекта.

Чаще всего, но далеко не всегда, объектом томографического исследования являются живые ткани. На рисунке ниже представлен слой томограммы головы.



Данные томографии представляют собой трехмерный массив вокселей - элементов трехмерной регулярной сетки. Каждый воксел содержит одно значение плотности, как правило, типа short или ushort. Значения плотностей чаще всего лежат в диапазоне от 0 до 4096 или от -1000 до 3096.

Для перевода значения плотности в цвет используется передаточная функция – Transfer Function (TF). Выходные значения передаточной функции могут быть серыми, от черного до белого, или цветными, линейными или нелинейными.



В данной лабораторной работе будет использоваться линейная передаточная функция от черного к белому, все значения рассчитываются по формуле:

$$intensity = \frac{density - min\_density}{max\_density - min\_density} * 255$$

Каналы RGB выходного цвета будут равны рассчитанной интенсивности $R = G = B = intensity$.

1. Создание проекта

Для решения задачи послойной визуализации томограммы мы будем использовать язык C++ и стандарт и технологию OpenGL.

Создайте новый проект "QT GUI Application" на языке C++, дайте ему название <Фамилия>_tomogram_visualizer.

Главным классом программы будет класс View, отнаследованный от QGLWidget.

2. Чтение файла томограммы

Обычно файлы томограмм хранятся в файлах формата DICOM, но для простоты задачи в данной работе будет использоваться томограмма, сохраненная в бинарном формате (Рисунок 1).

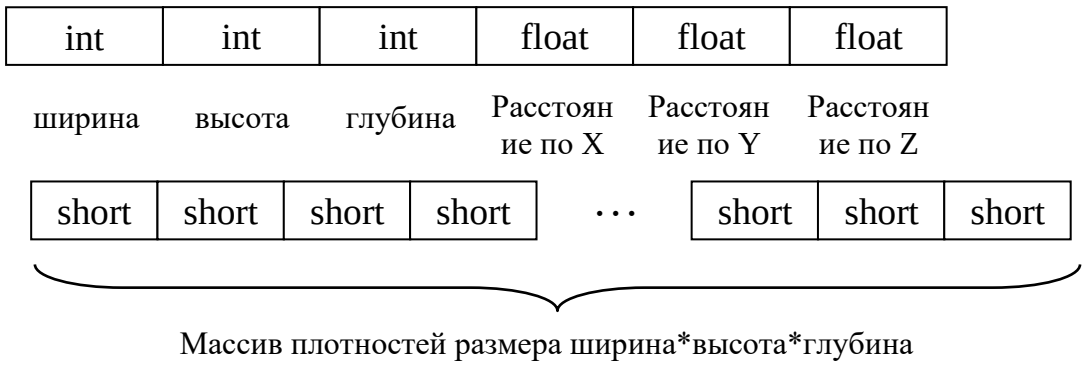


Рисунок 1. – Формат бинарных данных.

Для загрузки томограммы потребуется прочитать из бинарного файла размеры томограммы (3 числа в формате int), ширину, высоту и глубину одного вокселя (3 числа типа float) и массив данных типа short. Создайте класс для чтения данных из файла, назовите его Data.

Класс должен реализовывать следующие функции:

```
short getMin() – вернуть минимальное значение плотности, встречающейся в данных
short getMax() – вернуть максимальное значение плотности, встречающейся в данных

int getWidth()
int getHeight()
int getDepth()

int readFile(QString fileName) – прочитать файл и вычислить минимум и максимум данных
```

3. Классы для визуализации

Теперь займемся реализацией класса View. View отнаследован от класса QGLWidget. QGLWidget предоставляет функциональные возможности для

отображения графики OpenGL, интегрированной в приложение Qt. Класс-наследник QGLWidget можно использовать, как и любой другой QWidget, за исключением того, что теперь есть выбор между использованием QPainter и стандартными командами рендеринга OpenGL.

Примечание. Этот класс является частью устаревшего модуля Qt OpenGL, и, как и другие классы QGL, его следует избегать в новых приложениях. Вместо этого, начиная с Qt 5.4, нужно использовать классы QOpenGLWidget и QOpenGL.

QGLWidget предоставляет три удобные виртуальные функции, которые необходимо переопределить в своем подклассе для выполнения типичных задач OpenGL:

paintGL() – визуализирует сцену OpenGL. Вызывается всякий раз, когда виджет должен быть обновлен,

resizeGL() – устанавливает область просмотра OpenGL, проекцию и т. д. Вызывается при каждом изменении размера виджета (а также при его первом отображении, поскольку все вновь созданные виджеты получают событие изменения размера автоматически).

initializeGL() – Вызывается один раз перед первым вызовом resizeGL() или paintGL(). Устанавливает контекст рендеринга OpenGL, определяет списки отображения, здесь необходимо вызывать функции глобальных настроек построения изображения, которые нет необходимости указывать при построении кадра.

Класс-наследник QGLWidget автоматически вызывает методы:

При запуске: initializeGL() → resizeGL() → paintGL()

При изменении размера окна: resizeGL() → paintGL()

updateGL() вызывает paintGL()

4. Инициализация

В переопределенном методе initializeGL() устанавливаем заполняющий цвет, выбираем режим сглаживания GL_SMOOTH, в модельно-видовую матрицу записываем матрицу тождественного преобразования.

```
void View::initializeGL()
{
	// устанавливаем заполняющий цвет
	qglClearColor(Qt::white);
	// устанавливаем режим сглаживания
	glShadeModel(GL_SMOOTH);
	// задаем модельно-видовую матрицу
	glMatrixMode(GL_MODELVIEW);
	// загрузка единичной матрицы
	glLoadIdentity();
}
```

5. Изменение размеров окна

В переопределенном методе resizeGL() матрицу проекции сначала инициализируем матрицей тождественного преобразования (glLoadIdentity()). А затем задаём обращением к GL.Orto() ортогональное проецирование массива данных томограммы в окно вывода, которое попутно преобразует размеры массива в канонический видимый объем (CVV). Далее настраиваем окно обзора таким образом, чтобы оно было равно размеру синтезируемого изображения.

```
void View::resizeGL(int nWidth, int nHeight)
{
	// установка режима матрицы
	glMatrixMode(GL_PROJECTION);
	// загрузка единичной матрицы
	glLoadIdentity();
	// установка ортогонального преобразования
	glOrtho(0.0f, data.getWidth() - 1, 0.0f, data.getHeight() - 1, -1.0f, 1.0f);
	// установка окна обзора
	glViewport(0, 0, nWidth, nHeight);
	update();
}
```

4. Визуализация томограммы

В данной лабораторной работе будет проведено сравнение двух вариантов визуализации томограммы:

1. Отрисовка четырехугольниками, вершинами которых являются центры вокселей текущего слоя регулярной воксельной сетки. Цвет формируется на центральном процессоре и отрисовывается с помощью функции glBegin(GL_QUADS);
2. Отрисовка текстурой. Текущий слой томограммы визуализируется как один большой четырехугольник, на который изображение слоя накладывается как текстура аппаратной билинейной интерполяцией.

Всё, что отвечает за визуализацию должно находиться в функции paintGL(). Сначала необходимо очистить буферы экрана, потом вызвать функции, которые отвечают за отрисовку. У нас таких будет три: VisualizationQuads(), VizualizationTexture() и VisualizationQuadStrip() вы должны реализовать самостоятельно. Создайте прототипы функций в классе View.

```
void View::paintGL()
{
	qDebug() << "repaint" << visualization_state;
	// очистка экрана
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	switch (visualization_state)
	{
	case VISUALIZATION_QUADS:
		VisualizationQuads();
		break;
	case VISUALIZATION_QUADSTRIP:
		VisualizationQuadStrip();
		break;
```

```
	case VISUALIZATION_TEXTURE:
		VizualizationTexture();
		break;
	}
}
```

5. Создание передаточной функции (Transfer Function, TF)

TF - функция перевода значения плотностей томограммы в цвет. Диапазон визуализируемых плотностей называется окном визуализации. В нашем случае мы хотим, чтобы TF переводила плотности окна визуализации от минимальной встреченной в данных плотности до максимальной встреченной в данных плотности линейно в цвет от черного до белого (от 0 до 255).

```
QColor View::TransferFunction(short value)
{
	int c = (value - data.getMin()) * 255 / (data.getMax() - data.getMin());
	return QColor(c, c, c);
}
```

Подставляя вместо `data.getMin()` и `data.getMax()` другие параметры min и max, мы будет получать различные изображения для нашей томограммы. Часто TF имеет более сложную структуру, чем линейная зависимость от максимума и минимума, но в данной лабораторной работе нам достаточно такой.

6. Отрисовка томограммы четырехугольниками

Реализация функции VisualizationQuads(). Слой, который будет визуализироваться хранится в переменной layer (член класса View).

```
void View::VisualizationQuads()
{
	QColor c;
	int w = data.getWidth();
	int h = data.getHeight();
	for (int y = 0; y < (h - 1); y++)
		for (int x = 0; x < (w - 1); x++)
		{
			glBegin(GL_QUADS);

			c = TransferFunction(data[layer * w * h + y * w + x]);
			qglColor(c);
			glVertex2i(x, y);

			c = TransferFunction(data[layer * w * h + (y + 1) * w + x]);
			qglColor(c);
			glVertex2i(x, (y + 1));

			c = TransferFunction(data[layer * w * h + (y + 1) * w + x + 1]);
			qglColor(c);
			glVertex2i((x + 1), (y + 1));

			c = TransferFunction(data[layer * w * h + y * w + x + 1]);
			qglColor(c);
			glVertex2i((x + 1), y);
			glEnd();
```

```
        }
    }
```

Из томограммы извлекаются значения томограммы в 4 ячейках: $(x, y), (x + 1, y), (x, y + 1), (x + 1, y + 1)$. Эти значения заносятся в цвет вершин четырехугольника, и данный четырехугольник визуализируется. Данная операция происходит в цикле по ширине и высоте томограммы. Перечисление вершин четырехугольника происходит против часовой стрелки.

7. Перемотка слоев

Перемотку слоёв и другие взаимодействия с данными будем делать через механизм сигналов-слотов.
У нашего класса есть унаследованный от QWidget слот `keyPressEvent()`. Это значит, что эта функция вызывается тогда, когда пользователь нажимает на любую кнопку. Обратите внимание, что в обработчике используется функция nativeVirtualKey(), потому что функция key() зависит от включенной языковой раскладки.

```
void View::keyPressEvent(QKeyEvent* event)
{
	if (event->nativeVirtualKey() == Qt::Key_U)
	{
		// Подняться на слой выше
	}
	else if (event->nativeVirtualKey() == Qt::Key_D)
	{
		// Опуститься на слой ниже
	}
	else if (event->nativeVirtualKey() == Qt::Key_N)
	{
		// Переключиться на следующий тип рендеринга
	}
	// Подняться на слой выше
	update();
	}
```

8. Визуализация томограммы как текстуры

Суть визуализации томограммы, как текстуры будет заключаться в следующем: мы сделаем изображение из одного слоя данных на процессоре, передадим его в видеопамять а качестве текстуры, и будем производить текстурирование одного прямоугольника, заполняющего весь виджет. Генерация текстуры и загрузка в видеопамять выполняются один раз для слоя. Визуализация созданной текстуры происходит при переотрисовке окна.

9. Загрузка текстуры в память видеокарты

В классе View создайте переменную GLuint VBOtexture и функцию Load2dTexture(). Переменная VBOtexture будет хранить номер текстуры в памяти видеокарты. Функция glGenTextures генерирует уникальный номер

текстуры. Так как текстура в нашем приложении одна, то сгенерировать номер можно в initializeGL(), просто прописав строчку:

```
glGenTextures(1, &VBOtexture);
```

Функция glBindTexture() привязывает текстуру к определенному текстурному типу, делает ее активной, именно с помощью этой функции функция glTexImage2D() загружает текстуру в память видеокарты. Функция glTexParameteri() устанавливает параметры для текущей текстуры, привязанной к текстурному блоку.

Рассмотрим текстурирование поподробнее:

Когда выполняется следующий код:

```
GLuint tex;
glGenTextures(1, &tex);
```

Тогда создаётся пространство, которое будет содержать все данные, связанные с текстурой. И переменная 'tex' теперь хранит ссылку на эту текстуру.

Когда вызываются функции:

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex);
```

Объект текстуры 'tex' присоединяется к текстурному модулю номер 0. Вы говорите OpenGL, что объект текстуры 'tex' является 2D-текстурой, и вы будете ссылаться на него (пока он связан) с помощью GL_TEXTURE_2D. Обратите внимание, что в Qt нельзя гарантированно воспользоваться glActiveTexture, т.к. она не принадлежит к OpenGL 1.1.

Вызывая:

```
glTexImage(GL_TEXTURE_2D, ...);
```

Вы говорите OpenGL создать данные изображения, связать эти данные изображения с текстурой, которая в настоящее время привязана к активному текстурному модулю (GL_TEXTURE0) и привязана к блоку GL_TEXTURE_2D.

Таким образом, эта функция приведет к прикреплению данных изображения к текстурному объекту 'tex'. Но только потому, что это текстура в настоящее время привязана к текущему активному текстурному модулю и блоку GL_TEXTURE_2D.

Теперь, при вызове:

```
glBindTexture(GL_TEXTURE_2D, 0);
```

Вы говорите OpenGL, что текущий связанный текстурный объект ('tex') больше не привязан к текущему текстурному модулю и блоку GL_TEXTURE_2D. Это означает, что если вы снова вызовете glTexImage, это не повлияет на объект текстуры 'tex', до тех пор пока вы снова не свяжете 'tex' с текстурным блоком.

Короче говоря, 'tex' является автономным объектом памяти. Здесь хранятся все данные, связанные с текстурой. Открепление от текстурного блока не повлияет на содержимое объекта.

Вы можете повторять этот процесс столько, сколько хотите. Ну, пока OpenGL не исчерпает память. Вы можете создавать текстуры с помощью glGenTextures(), связывать их, создавать для них данные изображений и отменять привязку.

Теперь, когда приходит время использовать текстуры в качестве источника данных изображения, вы должны снова связать их. Но только те, которые вам нужны для этой конкретной операции рисования.

Допустим, у вас есть объект A и объект B. Для объекта A требуется текстура 'texA', связанная с текстурным модулем 0, а для объекта B нужны текстуры 'texB' и 'texC', связанные с модулями текстуры 0 и 1 соответственно.

Чтобы визуализировать объект A, вы делаете это:

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texA);
glEnable(GL_TEXTURE_2D); //Only use this if not using shaders.

glDrawElements(...);

//Cleanup texture binds.
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, 0);
glEnable(GL_TEXTURE_2D); //Only use this if not using shaders.
```

Чтобы визуализировать объект B, вы делаете это:

```
glActiveTexture(GL_TEXTURE0 + 0);
glBindTexture(GL_TEXTURE_2D, texB);
glEnable(GL_TEXTURE_2D); //Only use this if not using shaders.
glActiveTexture(GL_TEXTURE0 + 1);
glBindTexture(GL_TEXTURE_2D, texC);
glEnable(GL_TEXTURE_2D); //Only use this if not using shaders.

glDrawElements(...);

//Cleanup texture binds.
glActiveTexture(GL_TEXTURE0 + 0);
glBindTexture(GL_TEXTURE_2D, 0);
glEnable(GL_TEXTURE_2D); //Only use this if not using shaders.
glActiveTexture(GL_TEXTURE0 + 1);
```

```
	glBindTexture(GL_TEXTURE_2D, 0);
	glEnable(GL_TEXTURE_2D); //Only use this if not using shaders.
```

Несмотря на то, что привязка текстуры к текстурному блоку необходима для ее создания и использования, это не означает, что текстурные блоки и объекты текстуры постоянно связаны друг с другом.

Если бы мы хотели, при рендеринге objectB, 'texA' мог бы быть связан с текстурным блоком 1 вместо 'texC'.

Текстурный блок (texture target) и текстурный модуль (texture unit). Каждый текстурный модуль имеет несколько текстурных блоков, относящихся к разным видам текстур (например, GL_TEXTURE_2D, GL_TEXTURE_3D, и т.д.). Функции, которые используют текстурный блок для идентификации текстуры, работают с текстурой, привязанной к этому блоку (активного текстурного модуля). Начиная с OpenGL 4.5, большинство функций, которые идентифицируют объекты (например, текстуры, буферы) через цель, имеют альтернативные функции, которые позволяют ссылаться на объект напрямую через его имя (дескриптор). Альтернативы имеют префикс «Texture» в названии, где оригинальная функция начинается с «Tex» (например, glTextureParameter, а не glTexParameter).

Можно сказать, что в OpenGL доступен массив структур, где размер массива – это количество текстурных модулей, а структура содержит параметры для каждого блока, то есть каждый модуль текстуры может содержать одну текстуру каждого типа (блок – текстура определенного типа). Каждый текстурный блок может ссылаться только на одно изображение, содержать одну текстуру.

Ситуация с текстурами – исторический артефакт. С современными OpenGL редко можно связать несколько текстур с одним текстурным модулем, т.к. шейдер не может получить доступ к нескольким блокам в пределах одного текстурного модуля. Функции прямого доступа к состоянию, которые были добавлены в версии 4.5, в значительной степени устраняют необходимость их использования.

```
void View::Load2DTexture()
{
	glBindTexture(GL_TEXTURE_2D, VBOtexture);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, textureImage.width(), textureImage.height(),
		     0, GL_BGRA, GL_UNSIGNED_BYTE, textureImage.bits());
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	}
```

10. Отрисовка текстурированного прямоугольника

В классе View создайте переменную QImage textureImage и функцию genTextureImage(), которая будет генерировать изображение из томограммы при помощи созданной Transfer Function.

```
void View::genTextureImage()
```

```
{
	int w = data.getWidth();
	int h = data.getHeight();

	textureImage = QImage(w, h, QImage::Format_RGB32);

	for (int y = 0; y < h; y++)
		for (int x = 0; x < w; x++)
		{
			QColor c = TransferFunction(data[layer * w * h + w * y + x]);
			textureImage.setPixelColor(x, y, c);
		}
}
```

Функция VizualizationTexture() рисует один прямоугольник с наложенной текстурой.

```
void View::VizualizationTexture()
{
	glBegin(GL_QUADS);
	qglColor(QColor(255, 255, 255));

	glTexCoord2f(0, 0);
	glVertex2i(0, 0);

	glTexCoord2f(0, 1);
	glVertex2i(0, data.getHeight());

	glTexCoord2f(1, 1);
	glVertex2i(data.getWidth(), data.getHeight());

	glTexCoord2f(1, 0);
	glVertex2i(data.getWidth(), 0);

	glEnd();
}
```

Сделайте возможность переключаться между режимами визуализации четырехугольниками и текстурой по нажатию клавиши N (Next).

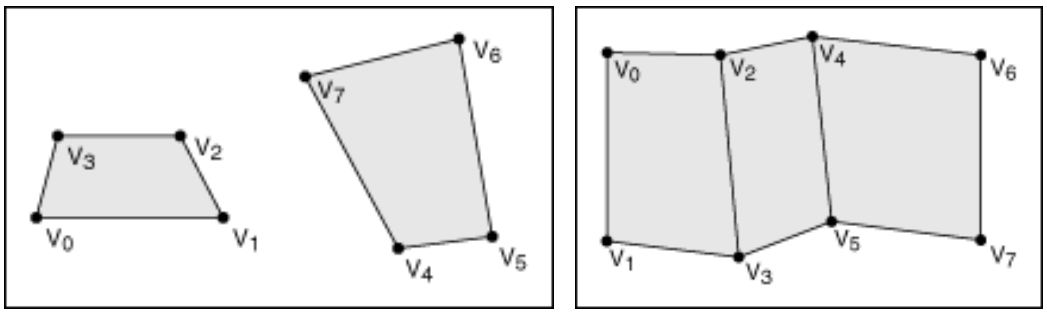
11. Задания для самостоятельной работы

1. Изменение Transfer Function

Создайте диалоговое окно, позволяющее задать новые минимальное и максимальное значение для линейной передаточной функции. Не забудьте, что при изменении TF в режиме рисования текстурой необходимо загружать новую текстуру в видеопамять.

2. Отрисовка при помощи QuadStrip

В OpenGL есть тип визуализации QuadStrip, когда первый четырехугольник рисуется 4 вершинами, а последующие - 2 вершинами, присоединенными к предыдущему четырехугольнику (рис. ниже). Таким образом для отрисовки N четырехугольников требуется не 4*N вершин, а 2*N+2 вершин, что положительно сказывается на скорости работы программы.



3. Реализуйте возможность загрузки других тестовых данных.
4. Реализуйте возможность перемотки слоёв по другой оси, чтобы можно было посмотреть данные в другом разрезе

12. Ссылки

1. https://habr.com/post/310790/- большой цикл статей по OpenGL.
2. https://www.intuit.ru/studies/courses/2313/613/lecture/13296 - ИНТУИТ. Создание графических моделей с помощью Open Graphics Library.