1. Overview Structure

Application chia thành nhiều module. Mỗi module đảm nhiệm một nhiệm vụ khác nhau.

Tất cả module đều bắt buộc phải có file module chính kế thừa từ Module Interface

class ModuleInterface

{

public:

virtual bool *hasGraphics*() = 0;

virtual GraphicsComposite\* *getGraphic*() = 0;

virtual bool *hasAction*() = 0;

virtual ActionInterface\* *getAction*() = 0;

virtual bool *loadOnBoot*() = 0;

virtual QString *getModuleName*() = 0;

virtual void *initModule*() = 0;

virtual void *terminalModule*() = 0;

};

Module có thể có Graphics, Action. Tùy đặc thù của Module. Nếu sử dụng Graphics thì phải kế thừa từ abstract class GraphicsComposite và phải khai báo với hệ thống (method hasGraphics return true).

class GraphicsComposite

{

protected:

GraphicsComposite\* graphicsMain;

vector<GraphicsComposite\*> graphics;

ActionListener\* actions;

vector<Menu\*> menus;

const int GRAPHICS\_ID = -1;

DemInterface\* demObject;

public:

void setMainGraphics(GraphicsComposite\*);

GraphicsComposite\* getMainGraphics();

void setActionPerform(ActionListener\* action);

ActionInterface\* getAction(QString);

void registerGraphics(GraphicsComposite\* graphic);

int addToGraphicsComponent();

bool addMenu(QString menuName, QAction \*action);

bool registerMenu(QMenuBar\*);

virtual void *setSize*(int width, int height);

virtual void *updateGraphics*();

virtual void *updatePaintGL*();

QAction\* createQAction( QString name );

virtual void *addVertex*(Vertex vertex, int col, int row);

void setDemObject(DemInterface\* dem);

virtual QSize *getSize*();

virtual void *mousePressEvent*(QMouseEvent \*event);

virtual void *mouseMoveEvent*(QMouseEvent \*event);

virtual void *mouseDoubleClickEvent*(QMouseEvent \*event);

virtual void *initial*() = 0;

virtual void *initializeGL*() = 0;

virtual void *paintGL*() =0 ;

virtual void *resizeGL*(int width, int height) = 0;

};

Abstract Graphics Composite cung cấp sẵn các hàm căn bản để Module có thể tương tác với hệ thống. Vd như để tạo một menu mới, Module chỉ cần gọi hàm bool addMenu(QString menuName, QAction \*action); là được.

Tương tự, nếu sử dụng Action thì phải kế thừa từ Action Interface & khai báo với hệ thống.

class ActionInterface

{

private:

ActionListener\* actionPerform;

public:

virtual void *initAction*() = 0;

virtual void *terminalAction*() = 0;

virtual QString *getActionName*() = 0;

virtual bool *loadOnBoot*() = 0;

virtual bool *requireDemObject*() = 0;

virtual void *setDemObject*(DemInterface\*) = 0;

virtual DemInterface\* *getDemObject*()=0;

virtual void *setGraphics*(GraphicsComposite\*)=0;

virtual void *setActionPerform*(ActionListener\*) =0;

};

Đối với các Module có sử dụng dữ liệu DEM thì bắt buộc phải có Action. Trong Action phải xin cấp phép sử dụng dữ liệu DEM (hàm requireDemObject trả về true) và hệ thống sẽ tự cấp dữ liệu DEM thông qua DemInterface.

1. Modules
   1. Module Files

Module Files sẽ tạo menu chọn file DEM. Read file DEM & generate mesh

Sử dụng thư viện GDAL để đọc file DEM => Raster data

Raster data là 1 mảng 1 chiều kiểu float chứa dữ liệu độ cao của từng pixel. Dựa vào Raster data và số cols, rows của file DEM, ta có 1 mảng 2 chiều:

Cols

Height

Rows

Từ dữ liệu mảng 2 chiều này, chúng ta quy đổi nó sang hệ quy chiếu dùng trong OpenGL với tâm là gốc tọa độ

1.0

1.0

0.0

-1.0

1.0

-1.0

-1.0

Giả sử ta cần tính tọa độ x, y từ mảng 2 chiều sang hệ quy chiếu dùng trong OpenGL:

NewX = (X – CenterX)/CenterX

NewY = (CenterY - Y)/CenterY

Với X,Y là tọa độ mảng 2 chiều

NewX, NewY : Tọa độ hệ quy chiếu trong OpenGL

CenterX : cols/2

CenterY: rows/2

Code:

for (int k = 0; k < rows; k=k+step) {

posY = (float)(centerY - k)/centerY;

for (int j =0; j < cols; j=j+step) {

posX = (float)(j - centerX)/centerX;

Đối với Z, Độ cao trong mảng 2 chiều sẽ nằm trong khoảng -11000 -> 9000. Tuy nhiên, 9000 thì khoảng cách quá lớn nên khi vẽ 3D khó phân biệt vị trí cao thấp. Do đó để mức trung bình : 6000.

NewZ = Data2D[X,Y]/6000;

Sau khi xác định xong tọa độ trong hệ quy chiếu của OpenGL, chúng ta tiến hành nội suy 3D bằng cách sử dụng triangle:

Từ 1 điểm bất kỳ, tìm 4 điểm liền kề với nó [(y+1, x), (y, x +1)] và [(y-1, x), (y, x-1)]. Tạo thành 2 tam giác. Cứ tiếp tục như vậy cho tới hết mảng 2 chiều.

Xác định vector normal:

Vector normal dùng để xác định hướng của ánh sáng khi chiếu vào địa hình. Từ đó xác định điểm sáng, tối trên địa hình.

* Cần phải xác định vector normal của từng vertex trên mesh. Khi xác định 3 điểm để vẽ triangle, chúng ta cũng tính luôn vector normal của triangle đó. Sau đó dùng vector normal này để gán cho các vertex của triangle.
* Khi 2 triangle cùng sử dụng chung 1 vertex, vector normal của vertex đó sẽ là vector normal của 2 triangle gộp lại

void DemObject::*addVertex*(Vertex vertex, int position)

{

int currentPosition = vertexs.size() - 1;

if (position > currentPosition) {

vertexs.push\_back(vertex);

} else {

vertexs[position].setNormal(QVector3D::normal(vertex.normal(), vertexs[position].normal()));

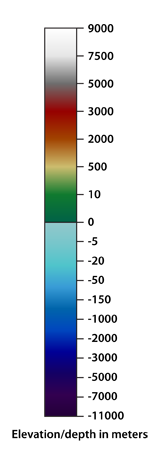
}

ind.push\_back(position);

}

* 1. Module Color

Tạo ra các dải màu dựa theo độ cao của địa hình. Module Files sẽ dùng các dải màu này để tô màu cho mesh



Source: <http://noaa.maps.arcgis.com/home/item.html?id=feb3c625dc094112bb5281c17679c769>

Tại mỗi vertex chúng ta có height của vertex đó. Map height này với màu như hình trên ta được màu của từng vertex.

* 1. Module Base Support

Tạo ra khung bao quanh mesh

Tìm kiếm tất cả các vertex nằm ở các hướng left, right, bottom & top của mesh. Dựa vào các vertex đó, vẽ triangle xuống điểm có cùng tọa độ nhưng có height = min(height) - 100

* 1. Module Rotate

Bổ sung tính năng xoay DEM

Module Rotate sẽ bắt sự kiện click chuột phải & move chuột của user, Xác định khoảng cách từ vị trí click và sau khi move

Dùng hàm glRotatef của OpenGL để rotate

glRotatef(xRot / 16.0, 1.0, 0.0, 0.0);

glRotatef(yRot / 16.0, 0.0, 1.0, 0.0);

glRotatef(zRot / 16.0, 0.0, 0.0, 1.0);

* 1. Module Zoom

Module Zoom bắt sự kiện xoay chuột giữa để tăng hoặc giảm mức độ zoom.

Dùng hàm glOtho và glViewport để zoom

int side = qMin(width, height);

glViewport(((width - side) / 2)\*zoomByScale, ((height - side) / 2)\*zoomByScale, side\*zoomByScale, side\*zoomByScale);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

float aspectRatio = (float)width/(float)height;

glOrtho(-aspectRatio\*zoomByScale, aspectRatio\*zoomByScale, -1\*zoomByScale, 1\*zoomByScale, 1.0, 15.0);

glMatrixMode(GL\_MODELVIEW);

* 1. Module Move

Module Move bắt sự kiện click chuột trái & move chuột của user để xác định hướng mà user đang muốn move tới

int distanceX = event->x() - lastPos.x();

int distanceY = event->y() - lastPos.y();

if (abs(distanceX) < abs(distanceY)) {

y -= distanceY \* 0.0001;

} else {

x += distanceX \* 0.0001;

}