# OsgEarth中设置模型运动路径，并绘制雷达扫描、动态实时绘制运动轨迹、跟随彩带

工程结构

- #BuildRader/BuildRaderCallback 主要用来处理雷达相关操作
- #CreateTrackCallback 主要是用来实时绘制飞机历史轨迹
- #TrailerCallback 主要是飞机飞行彩带
- #ViewerWidget 主要是窗体相关，并初始化

## 需求

首先我们在场景中加载一个机场模型、一个飞机模型，将飞机沿着跑道方向放置于机场之中，根据我们设置的关键点的位置信息飞机，飞行过程中有雷达扫描飞机下方，飞机尾带跟随飞机运动，并实时绘制飞机的历史航迹。

## 加载机场和飞机

```
void CViewerWidget::addAirport()
{
 m_rpCoordSystem = new osg::CoordinateSystemNode;   //创建坐标系节点
 m_rpCoordSystem->setEllipsoidModel(new osg::EllipsoidModel());   //设置设置椭圆体模型
 //加载机场
 m_rpnodeAirport = osgDB::readNodeFile("D:/Code/src/model/airport.ive");
 m_rpmtAirport = new osg::MatrixTransform;
 m_rpmtAirport->addChild(m_rpnodeAirport);
 m_rpRoot->addChild(m_rpmtAirport);
 //设置机场矩阵
 osg::Matrixd mtTemp;
 m_rpCoordSystem->getEllipsoidModel()->computeLocalToWorldTransformFromLatLongHeight(osg::DegreesToRadians(34.3762), osg::DegreesToRadia
ns(109.1263),460 , mtTemp);
 m_rpmtAirport->setMatrix(mtTemp);   //根据经、维、高得到飞机场想要的矩阵
 //加载飞机
 m_rpnodeAirFly = osgDB::readNodeFile("D:/Code/src/model/B737.ive");
 m_rpmtFlyself = new osg::MatrixTransform;
 m_rpmtFlyself->setMatrix(osg::Matrixd::scale(10, 10, 10)*osg::Matrixd::rotate(3*osg::PI_4,osg::Vec3(0, 0, 1)));

 m_rpmtFlyself->getOrCreateStateSet()->setMode(GL_RESCALE_NORMAL, osg::StateAttribute::ON);   //设置属性，光照法线
 m_rpmtFlyself->addChild(m_rpnodeAirFly);

 m_rpmtFlyself->addChild(m_pBuildRader->BuildRader(500,300).get());
 m_rpmtFly = new osg::MatrixTransform;
 m_rpmtFly->addChild(m_rpmtFlyself);
 m_rpRoot->addChild(m_rpmtFly);
 //设置飞机矩阵
 m_rpCoordSystem->getEllipsoidModel()->computeLocalToWorldTransformFromLatLongHeight(osg::DegreesToRadians(34.3834), osg::DegreesToRadia
ns(109.1347), 537, mtTemp);
 m_rpmtFly->setMatrix(mtTemp);
}
```

## 创建飞机历史航迹

```
void CViewerWidget::BuildHistoryRoute(osg::MatrixTransform* scaler ,float lineWidth)
{
 osg::ref_ptr<osg::Group> rpgroup = new osg::Group;

 scaler->addUpdateCallback(new CreateTrackCallback(rpgroup,scaler,lineWidth));
 m_rpRoot->addChild(rpgroup);
}
```

## 设置飞机视角跟随 并设置飞行路径

```
void CViewerWidget::DoAPreLine()
{
 osg::ref_ptr<osg::Vec4Array> rpvaTemp = new osg::Vec4Array;

 rpvaTemp->push_back(osg::Vec4(109.1347, 34.3834, 537, 50));
 rpvaTemp->push_back(osg::Vec4(109.1174, 34.3686, 567, 500));

 rpvaTemp->push_back(osg::Vec4(109.1173, 34.3685, 566, 800));
 rpvaTemp->push_back(osg::Vec4(108.8794, 34.1944, 3000, 800));
 rpvaTemp->push_back(osg::Vec4(107.1302, 34.3941, 5000, 500));
 rpvaTemp->push_back(osg::Vec4(108.9387, 34.9202, 8000, 200));
 rpvaTemp->push_back(osg::Vec4(109.5066, 34.51, 3000, 200));
 rpvaTemp->push_back(osg::Vec4(109.1347, 34.3834, 537, 200));

 m_rpAnimationPath = CreateAirPath(rpvaTemp);
}

void CViewerWidget::DoPreLine()
{
 m_rpmtFly->setUpdateCallback(new osg::AnimationPathCallback(m_rpAnimationPath, 0.0, 1.0));
 //设置视角跟踪
 m_rpEarthManipulator->setViewpoint(osgEarth::Viewpoint(109.1347, 34.3834,0,24.261,-21.6,1000),5);
 m_rpEarthManipulator->setTetherNode(m_rpnodeAirFly);
}
```

## 创建飞机飞行彩带

```
void CViewerWidget::BuildRibbon( int size, osg::MatrixTransform* scaler ,int ribbonWidth)
{
 osg::ref_ptr<osg::Geometry> rpgeom = new osg::Geometry;
 //设置顶点
 osg::ref_ptr<osg::Vec3Array> rpvec3Vertex = new osg::Vec3Array(size);
 //设置颜色
 osg::ref_ptr<osg::Vec4Array> rpvec4Color = new osg::Vec4Array(size);

 for(unsigned int i = 0;i <size-1;i+=2)
 {
  (*rpvec3Vertex)[i] = osg::Vec3(0,0,0);
  (*rpvec3Vertex)[i+1] = osg::Vec3(0,0,0);

  float falpha = sinf(osg::PI * (float)i / (float)size);

  (*rpvec4Color)[i] = osg::Vec4(m_vec3RibbonColor,falpha);
  (*rpvec4Color)[i+1] = osg::Vec4(m_vec3RibbonColor,falpha);
 }
 //场景数据动态改变
 rpgeom->setDataVariance(osg::Object::DYNAMIC);
 //禁用显示列表，动态更新不安全
 rpgeom->setUseDisplayList(false);
 //使用VBO模式
 rpgeom->setUseVertexBufferObjects(true);

 rpgeom->setVertexArray(rpvec3Vertex);

 rpgeom->setColorArray(rpvec4Color);
 rpgeom->setColorBinding(osg::Geometry::BIND_PER_VERTEX);

 rpgeom->addPrimitiveSet(new osg::DrawArrays(GL_QUAD_STRIP,0,size));

 osg::ref_ptr<osg::Geode> rpgeode = new osg::Geode;
 rpgeode->addDrawable(rpgeom);
 //灯光、透明度
 rpgeom->getOrCreateStateSet()->setMode(GL_LIGHTING,osg::StateAttribute::OFF);
 rpgeom->getOrCreateStateSet()->setMode(GL_BLEND,osg::StateAttribute::ON);
 rpgeom->getOrCreateStateSet()->setRenderingHint(osg::StateSet::TRANSPARENT_BIN);

 scaler->addUpdateCallback(new CTrailerCallback(rpgeom,size,ribbonWidth));

 m_rpRoot->addChild(rpgeode);
}
```

## 计算飞机飞行姿态

```
//根据输入的控制点，输出一个路径，格式为（经、纬、高、速）
osg::AnimationPath* CViewerWidget::CreateAirPath( osg::Vec4Array* ctrl )
{
 osg::ref_ptr<osg::AnimationPath> rpAnimationPath = new osg::AnimationPath;
 rpAnimationPath->setLoopMode(osg::AnimationPath::NO_LOOPING);
 double dshuiPingAngle= 0.0;    //水平方向需要转的角度
 double dchuiZhiAngle = 0.0;    //垂直方向需要转的角度
 double time = 0;

 osg::Matrix matrix;
 //当前点
 osg::Vec3d Vec3positionCur;
 //下一点
 osg::Vec3d Vec3positionNext;

 for(osg::Vec4Array::iterator iter = ctrl->begin(); iter != ctrl->end(); iter++)
 {
  //下一个点
  osg::Vec4Array::iterator iter2 = iter;
```

```
 iter2 ++;

 //如果只有两个点
 if(iter2 == ctrl->end())
 {
  break;
 }
 //将经纬度转换到世界坐标系里面 为了计算竖直方向的坐标
 double x, y, z;
 m_rpCoordSystem->getEllipsoidModel()->convertLatLongHeightToXYZ(osg::DegreesToRadians(iter->y()), osg::DegreesToRadians(iter->x()), iter->z(), x,
y, z);
 Vec3positionCur = osg::Vec3(x, y, z);
 m_rpCoordSystem->getEllipsoidModel()->convertLatLongHeightToXYZ(osg::DegreesToRadians(iter2->y()), osg::DegreesToRadians(iter2->x()), iter2->z(
), x, y, z);
 Vec3positionNext = osg::Vec3(x, y, z);

 //求出水平夹角 经度相同
 if(iter->x() == iter2->x())
 {
  dshuiPingAngle = osg::PI_2;
 }
 else
 {
  dshuiPingAngle = atan((iter2->y() - iter->y())/(iter2->x() - iter->x()));
  if(iter2->x() > iter->x())
  {
   dshuiPingAngle += osg::PI;
  }
 }

 //求垂直夹角 高度一致
 if(iter->z() == iter2->z())
 {
  dchuiZhiAngle = 0;
 }
 else
 {
  //经纬度一致，高度不一致
  if(0 == sqrt(pow(dGetDis(Vec3positionCur, Vec3positionNext), 2)) - pow((iter2->z() - iter->z()), 2))
  {
   dchuiZhiAngle = osg::PI_2;
  }
  else
  {
   //求出高度差
   dchuiZhiAngle = atan((iter2->z() - iter->z()) /  sqrt(pow(dGetDis(Vec3positionCur, Vec3positionNext), 2)) - pow((iter2->z() - iter->z()), 2) );
  }
  if(dchuiZhiAngle>=osg::PI_2)
  {
   dchuiZhiAngle = osg::PI_2;
  }
  if(dchuiZhiAngle <= -osg::PI_2)
  {
   dchuiZhiAngle = -osg::PI_2;
  }
 }
 //求飞机的变换矩阵
 m_rpCoordSystem->getEllipsoidModel()->computeLocalToWorldTransformFromLatLongHeight(osg::DegreesToRadians(iter->y()), osg::DegreesToRadia
ns(iter->x()), iter->z(), matrix);
 m_quatRotation.makeRotate(0, osg::Vec3(1.0, 0.0, 0.0), dchuiZhiAngle+osg::PI_2, osg::Vec3(0.0, 1.0, 0.0), dshuiPingAngle-osg::PI_4, osg::Vec3(0.0, 0.0
, 1.0));
 matrix.preMultRotate(m_quatRotation);

 rpAnimationPath->insert(time, osg::AnimationPath::ControlPoint(Vec3positionCur, matrix.getRotate()));
```

```
 //把下一个点的时间求出来
 time += dGetRunTime(Vec3positionCur, Vec3positionNext, iter2->w());
}
//只有两个点时
rpAnimationPath->insert(time, osg::AnimationPath::ControlPoint(Vec3positionNext, matrix.getRotate()));

return rpAnimationPath.release();
}
double CViewerWidget::dGetRunTime( osg::Vec3d from,osg::Vec3d to,double speed )
{
if(speed == 0)
{
 return 1000000000;
}
else
{
 return dGetDis(from,to)/speed;
}
}

double CViewerWidget::dGetDis( osg::Vec3d from,osg::Vec3d to )
{
return std::sqrt( (to.x() - from.x())*(to.x() - from.x()) +  (to.y() - from.y())*(to.y() - from.y()) + (to.z() - from.z())*(to.z() - from.z()));
}
```

其中飞机飞行姿态的计算，如何从当前点到下一个点，这里输入的位置信息osg::Vec4Array* ctrl是经度、维度、高度、速度，在这里我们需要计算飞机机头左右的转向角和飞机机头向上、向下的俯仰角，为了方便理解，我画了一个草图

1. 其中，飞机在A点，下一个关键点在B点，首先，我们要明白飞机怎样才能飞到B点，首先，飞机机头要进行水平转向一定的角度和AC同向，然后向上垂直转向，和AB同向，这样，才可以按照航迹正确的飞到B点。
2. 首先我们计算飞机的水平转向的角度，角度1的tan值等于A B两点的维度差的值/经度差的值，其中要进行考虑，经度相同时，成90度直角。
3. 然后计算飞机的垂直转向角度，首先考虑，经纬度不一致，高度一致，然后考虑经纬度一致，高度不一致，也就是第二个点在第一个点正上方的特殊情况。角度2的tan值就等于A B 两点的高度差/距离。

## TrailerCallback.h

```h
#pragma once

#include <osgViewer/Viewer>
#include <osgEarth/MapNode>
#include <osg/AnimationPath>
#include <osgEarth/Utils>
#include <QBoxLayout>
#include <QTimer>
#include <QWidget>
#include <osgEarthUtil/EarthManipulator>
#include <osgParticle/FireEffect>
#include <osg/Geometry>
#include <osg/Geode>
#include <osg/ShapeDrawable>
#include <osgGA/GUIEventHandler>
#include <math.h>
#include <iostream>
#include <fstream>


class CTrailerCallback:public osg::NodeCallback
{
public:
 CTrailerCallback(osg::Geometry* ribbon, int size,int ribbonWidth);

 ~CTrailerCallback();

 virtual void operator()(osg::Node* node, osg::NodeVisitor* nv);

private:
 osg::observer_ptr<osg::Geometry> m_opGeometryRibbon;
 int m_nsize;
 int m_nwidth;
};
```

## TrailerCallback.cpp

```
#include "TrailerCallback.h"


CTrailerCallback::CTrailerCallback( osg::Geometry* ribbon, int size,int ribbonWidth )
{
 m_opGeometryRibbon = ribbon;
 m_nsize = size;
 m_nwidth = ribbonWidth;
}

void CTrailerCallback::operator()( osg::Node* node, osg::NodeVisitor* nv )
{
 osg::MatrixTransform* pmtTrans = dynamic_cast<osg::MatrixTransform*> (node);
 if(pmtTrans && m_opGeometryRibbon.valid())
 {
  osg::Matrix mtx = pmtTrans->getMatrix();
  osg::Vec3Array* pvec3Vertex = dynamic_cast<osg::Vec3Array*>(m_opGeometryRibbon->getVertexArray());

  for(unsigned int i = 0;i<m_nsize-3;i+=2)
  {
   (*pvec3Vertex)[i] = (*pvec3Vertex)[i+2];
   (*pvec3Vertex)[i+1] = (*pvec3Vertex)[i+3];
  }

  (*pvec3Vertex)[m_nsize-2] = osg::Vec3(0.0f,-m_nwidth,0.0f)* mtx;
  (*pvec3Vertex)[m_nsize-1] = osg::Vec3(0.0f,m_nwidth,0.0f)* mtx;

  pvec3Vertex->dirty();
  m_opGeometryRibbon->dirtyBound();
 }
 traverse(node,nv);
}

CTrailerCallback::~CTrailerCallback()
{

}
```

## CreateTrackCallbcak.h

```h
#pragma once

#include <osgViewer/Viewer>
#include <osgEarth/MapNode>
#include <osg/AnimationPath>
#include <osgEarth/Utils>
#include <QBoxLayout>
#include <QTimer>
#include <QWidget>
#include <osgEarthUtil/EarthManipulator>
#include <osgParticle/FireEffect>
#include <osg/Geometry>
#include <osg/Geode>
#include <osg/ShapeDrawable>
#include <osgGA/GUIEventHandler>
#include <math.h>
#include <iostream>
#include <fstream>
#include <osg/LineWidth>

//自定义轨道回调类
class CreateTrackCallback:public osg::NodeCallback
{
public:
 CreateTrackCallback(osg::Group* root,osg::MatrixTransform* scaler,float ribbonWidth);
 ~CreateTrackCallback();

 osg::ref_ptr<osg::Geode> BuildTrack(osg::Vec3 m_Vec3LatPoint,osg::Vec3 m_Vec3CurPoint);
 virtual void operator()(osg::Node* node,osg::NodeVisitor* nv);
 //上一帧模型位置坐标点
 osg::Vec3 m_Vec3LastPosition;
 //当前模型位置坐标点
 osg::Vec3 m_Vec3CurPosition;


 osg::observer_ptr<osg::Geometry> m_opGeometryRibbon;

 osg::ref_ptr<osg::MatrixTransform> m_rpmtFly;

 osg::Group* m_proot;

 int m_nsize;

 int m_nwidth;
};
```

## CreateTrackCallbcak.cpp

```cpp
#include "CreateTrackCallbcak.h"


CreateTrackCallback::CreateTrackCallback(osg::Group* root,osg::MatrixTransform* scaler,float lineWidth)
{
 m_proot = root;
 m_nwidth = lineWidth;
 m_rpmtFly = scaler;
}

CreateTrackCallback::~CreateTrackCallback( )
{
}

void CreateTrackCallback::operator()( osg::Node* node,osg::NodeVisitor* nv )
{
```

```
osg::MatrixTransform* pmtTrans = dynamic_cast<osg::MatrixTransform*> (node);
if(pmtTrans)
{
 osg::Matrix mtx = pmtTrans->getMatrix();
 m_Vec3CurPosition = mtx.getTrans();

 m_proot->addChild(BuildTrack(m_Vec3LastPosition,m_Vec3CurPosition));
}
traverse(node,nv);
m_Vec3LastPosition = m_Vec3CurPosition;

}

osg::ref_ptr<osg::Geode> CreateTrackCallback::BuildTrack(osg::Vec3 m_Vec3LatPoint,osg::Vec3 m_Vec3CurPoint)
{
osg::ref_ptr<osg::Geode> rpGeode = new osg::Geode;
osg::ref_ptr<osg::Geometry> rpGeom = new osg::Geometry;
osg::ref_ptr<osg::TessellationHints> rpHints = new osg::TessellationHints;
rpHints->setDetailRatio(0.5f);

//顶点数组
osg::ref_ptr<osg::Vec3Array> rpVec3Array = new osg::Vec3Array;
osg::ref_ptr<osg::Vec4Array> rpVec4Array = new osg::Vec4Array;

rpVec3Array->push_back(m_Vec3LatPoint);
rpVec3Array->push_back(m_Vec3CurPoint);


rpGeom->setVertexArray(rpVec3Array); //设置顶点
rpGeom->addPrimitiveSet(new osg::DrawArrays(osg::PrimitiveSet::LINES,0,rpVec3Array->size()));   //设置关联方式 线段

rpVec4Array->push_back(osg::Vec4f(1,0,0,1.0));
rpGeom->setColorArray(rpVec4Array);   //设置顶点颜色
rpGeom->setColorBinding(osg::Geometry::BIND_OVERALL); //设置关联方式

rpGeom->setDataVariance(osg::Object::DYNAMIC);

rpGeom->setUseVertexBufferObjects(true);

//设置线宽
osg::ref_ptr<osg::LineWidth> lw = new osg::LineWidth(m_nwidth);
rpGeom->getOrCreateStateSet()->setAttribute(lw, osg::StateAttribute::ON);
rpGeode->getOrCreateStateSet()->setMode(GL_LIGHTING,osg::StateAttribute::OFF);
rpGeom->getOrCreateStateSet()->setMode(GL_BLEND,osg::StateAttribute::ON);   //混合色
rpGeom->getOrCreateStateSet()->setRenderingHint(osg::StateSet::TRANSPARENT_BIN);   //透明度

rpGeode->addDrawable(rpGeom.get());
return rpGeode;
}
```

## BuildRader.h

```h
#pragma once

#include "BuildRaderCallback.h"
class CBuildRader
{
public:
 CBuildRader()
 {

 }
 ~CBuildRader()
 {

 }
public:
 //创建雷达圆锥图形
 osg::ref_ptr<osg::Geode> BuildRader(float fRadius, float fHeight);

 float m_fHeight;
 float m_fRadius;

 CBuildRaderCallback * buildRaderCallback;

};
```

## BuildRader.cpp

```cpp
#include "BuildRader.h"


osg::ref_ptr<osg::Geode> CBuildRader::BuildRader( float fRadius, float fHeight )
{
buildRaderCallback = new CBuildRaderCallback(2,fRadius,fHeight);

osg::ref_ptr<osg::Geode> rpGeode = new osg::Geode;
osg::ref_ptr<osg::Geometry> rpGeom = new osg::Geometry;
osg::ref_ptr<osg::TessellationHints> rpHints = new osg::TessellationHints;
rpHints->setDetailRatio(0.5f);


//顶点数组
osg::ref_ptr<osg::Vec3Array> rpVec3Array = new osg::Vec3Array;
osg::ref_ptr<osg::Vec4Array> rpVec4Array = new osg::Vec4Array;

rpVec3Array->push_back(osg::Vec3f(0,0,0));
rpVec3Array->push_back(osg::Vec3f(0,0,-fHeight));
rpVec3Array->push_back(osg::Vec3f(fRadius,0,-fHeight));

rpGeom->setVertexArray(rpVec3Array); //设置顶点
rpGeom->addPrimitiveSet(new osg::DrawArrays(osg::PrimitiveSet::TRIANGLES,0,rpVec3Array->size()));    //设置关联方式 三角形

rpVec4Array->push_back(osg::Vec4f(1,0,0,0.5));
rpGeom->setColorArray(rpVec4Array);    //设置顶点颜色
rpGeom->setColorBinding(osg::Geometry::BIND_PER_VERTEX); //设置关联方式

rpGeom->setDataVariance(osg::Object::DYNAMIC);

rpGeom->setUseVertexBufferObjects(true);

rpGeom->getOrCreateStateSet()->setMode(GL_LIGHTING,osg::StateAttribute::OFF);
//混合色
rpGeom->getOrCreateStateSet()->setMode(GL_BLEND,osg::StateAttribute::ON);
//透明度
rpGeom->getOrCreateStateSet()->setRenderingHint(osg::StateSet::TRANSPARENT_BIN);

rpGeode->addDrawable(rpGeom.get());
//扫描更新回调函数
rpGeode->addUpdateCallback(buildRaderCallback);
return rpGeode;
}
```

## BuildRaderCallback.cpp

```cpp
#include "BuildRaderCallback.h"


CBuildRaderCallback::CBuildRaderCallback( float fRotateSpeed,float fRotateRadius,float fRotateHeight )
{
 m_fSpeed = fRotateSpeed;
 m_fRadius = fRotateRadius;
 m_fHeight = fRotateHeight;
}

CBuildRaderCallback::~CBuildRaderCallback()
{

}

void CBuildRaderCallback::operator()( osg::Node* node,osg::NodeVisitor* nv )
{

 osg::Geode * pGeode = dynamic_cast<osg::Geode *>(node);
 osg::ref_ptr<osg::Geometry> rpGeo = dynamic_cast<osg::Geometry*> (pGeode->getDrawable(0));


 //获取顶点数组， 设置连接方式
 osg::ref_ptr<osg::Vec3Array> rpVertexArray = dynamic_cast<osg::Vec3Array*>(rpGeo->getVertexArray());


 double dRotateTime = nv->getFrameStamp()->getReferenceTime(); //获取当前运行时间

 rpVertexArray->push_back(rpVertexArray->at(0));
 rpVertexArray->push_back(rpVertexArray->at(1));
 rpVertexArray->push_back(osg::Vec3(m_fRadius * cosf(dRotateTime * m_fSpeed),m_fRadius * sinf(dRotateTime * m_fSpeed),-m_fHeight));
 rpVertexArray->erase(rpVertexArray->begin());
 rpVertexArray->erase(rpVertexArray->begin());
 rpVertexArray->erase(rpVertexArray->begin());

 rpVertexArray->dirty();

 //更新轨迹
 traverse(node,nv);
}
```

## BuildRaderCallback.h

```cpp
#pragma once

#include <osgViewer/Viewer>
#include <osgEarth/MapNode>
#include <osg/AnimationPath>
#include <osgEarth/Utils>
#include <QBoxLayout>
#include <QTimer>
#include <QWidget>
#include <osgEarthUtil/EarthManipulator>
#include <osgParticle/FireEffect>
#include <osg/Geometry>
#include <osg/Geode>
#include <osg/ShapeDrawable>
#include <osgGA/GUIEventHandler>
#include <math.h>
#include <iostream>
#include <fstream>

//自定义回调类
class CBuildRaderCallback:public osg::NodeCallback
{
public:
 CBuildRaderCallback(float fRotateSpeed,float fRotateRadius,float fRotateHeight);
 ~CBuildRaderCallback();
 virtual void operator()(osg::Node* node,osg::NodeVisitor* nv);
private:
 float m_fSpeed; //旋转速度
 float m_fRadius; //距（0，0，0）距离
 float m_fHeight;
};
```