Accurate, Dense, and Robust Multi-View Stereopsis论文分析与代码实现（二）

数据集的可视化

多视图数据集

开始讲论文之前，先谈谈论文实验所用的数据。这个数据可以从http://vision.middlebury.edu/mview/data/上下载。数据分为Temple和Dino。在整个论文的讲解过程中使用Temple数据。

Temple数据又分成3部分：

1. “Temple” data set：77 Mb、 312张
2. “TempleRing” data set：11 Mb、47张
3. “TempleSparseRing” data set： 4 Mb、16张

每张照片是640x480的分辨率。由于MVS算法的处理时间较长，因此测试的时候可以使用照片数量较少的TempleSparseRing数据。这些数据是在Stanford大学的Stanford Spherical Gantry拍摄的。这个设备看上去并不复杂，基于这个设备Stanford大学进行了很多实验，发

表了不少优秀论文。为什么我们不能做一个这样的实验设备呢?

每一项数据包含4类文件:

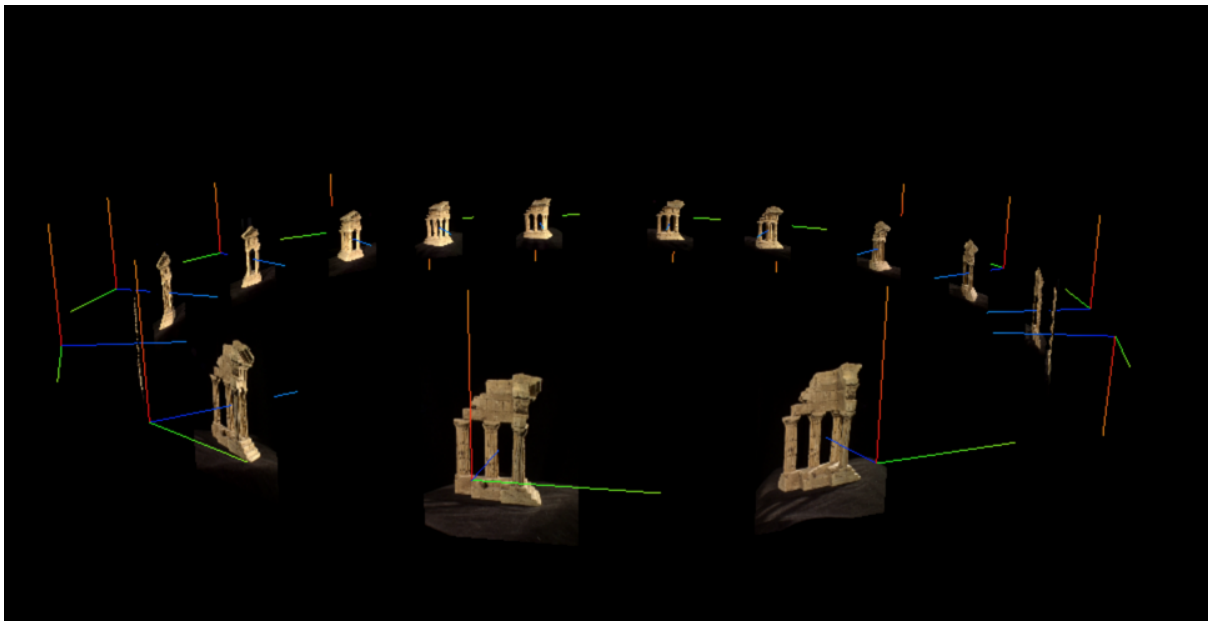
1. name*.png文件，图像数据。
2. name_ang.txt文件，拍摄图像的相机经纬度（相机分布在球面上）。
3. name_par.txt文件，相机参数。每一行的格式为：”imgname.png k11 k12 k13 k21 k22 k23 k31 k32 k33 r11 r12 r13 r21 r22 r23 r31 r32 r33 t1 t2 t3”，对应相机的投影矩阵$K\begin{bmatrix}R & t\end{bmatrix}$，其中$K$是$3\times3$的矩阵、$R$是$3\times3$的矩阵，$t$是$3\times1$的矩阵。
4. README.txt文件，拍摄物体的其他信息（例如包围盒的信息）。

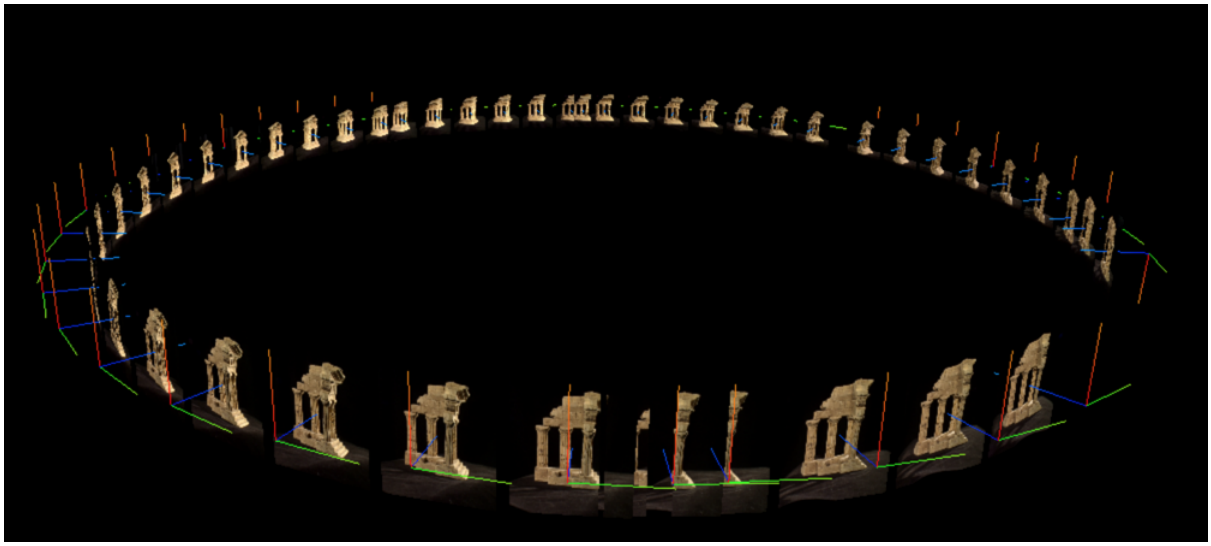
上述文件中name_par.txt最为重要，下面给出一个示例:

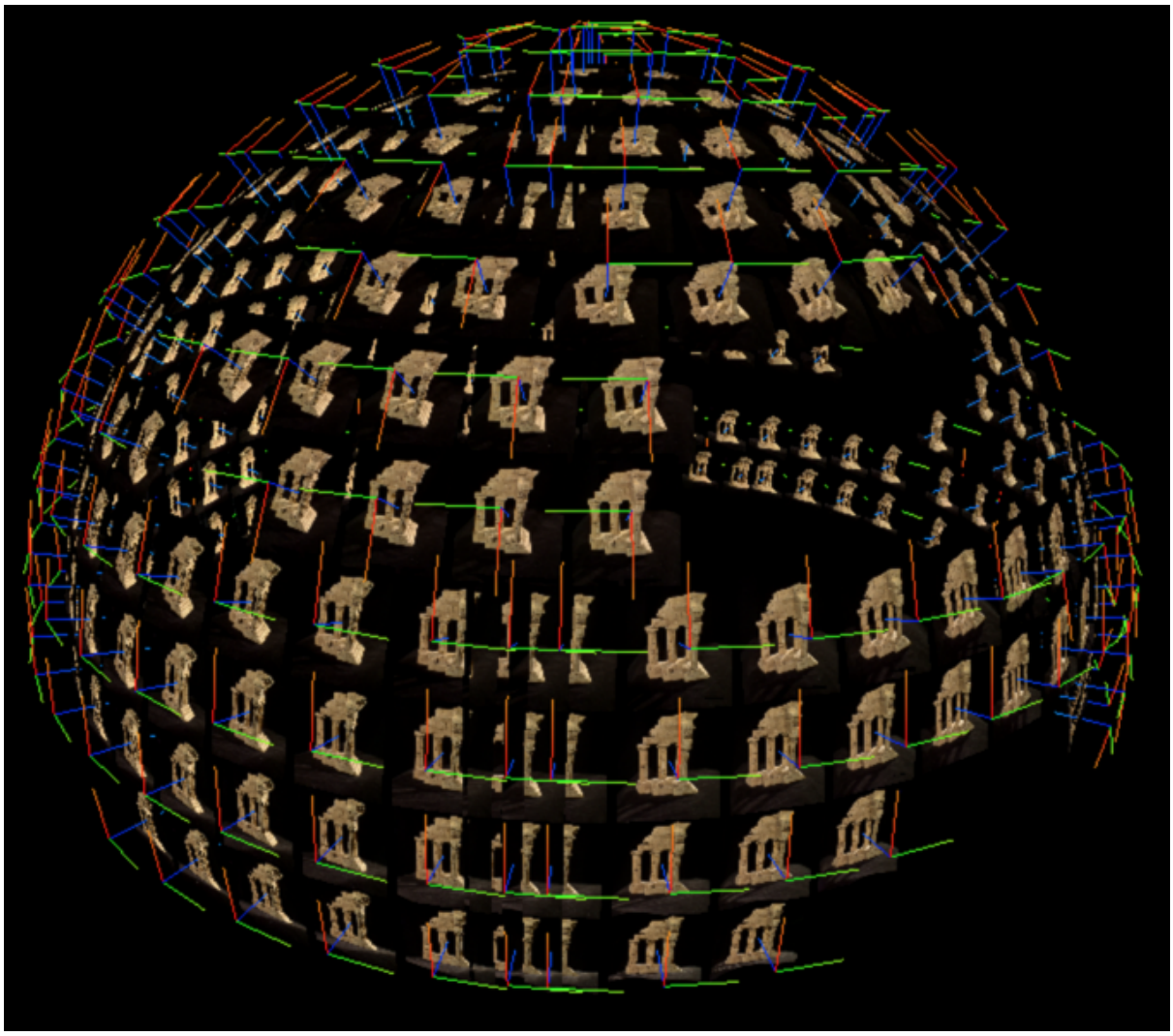
文件第一行为相片的个数（相机参数的数量、行数），第二行开始每一行对应一个相机参数。通过解析文件就可以构造每个相机所对应的投影矩阵（Projection Matrix）。

可视化效果

下图分别是templeSparseRing、templeRing和templ数据的展现效果，程序使用了WebGL技术。







上图中每个红、绿、蓝组成的坐标轴表示一个相机，其中红、绿、蓝分别对应相机的X轴、Y轴和Z轴。坐标轴前面是该相机拍摄的图像。

可视化原理

这个程序是怎么做的呢？先来看看基本原理。相机参数为$K\begin{bmatrix}R & t\end{bmatrix}$，其中$\begin{bmatrix}R & t\end{bmatrix}$表示了相机的朝向和位置。假设$\bar{X}$表示相机坐标系下的坐标，$X$表示世界坐标系下的坐标。则两者之间存在如下的转换关系：

$$\bar{X} = RX + t \tag{13}$$

用齐次坐标表示即为：

$$\begin{bmatrix}\bar{X} \\ 1\end{bmatrix} = \begin{bmatrix}R & t \\ 0 & 1\end{bmatrix}\begin{bmatrix}X \\ 1\end{bmatrix} \tag{38}$$

所以

$$X = R^{-1}(\bar{X} - t) = R^T(\bar{X} - t) = R^T\bar{X} - R^T t \tag{3}$$

或者齐次坐标形式

$$\begin{bmatrix} X \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \bar{X} \\ 1 \end{bmatrix} \tag{100}$$

$$= \begin{bmatrix} R^T & -R^T t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{X} \\ 1 \end{bmatrix} \tag{101}$$

其中$C = -R^T t$称为相机在世界坐标系中的位置。
在相机坐标系中，相机的X、Y、Z轴分别为$(1,0,0)^T$、$(0,1,0)^T$、$(0,0,1)^T$，则可以根据公式(???)或者(???)转换为世界坐标系，例如在齐次坐标系下：

$$\begin{bmatrix} R^T & -R^T t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{101}$$

该矩阵运算可以一次将相机坐标一次转换为世界坐标。

程序代码

程序使用Javascript编写，利用了3D库three.js，矩阵运算库sylvester.js和Javascript框架jquery.js。程序可以在谷歌浏览器和Safari浏览器上运行（其他浏览器未测试兼容性）。通常3D程序都是用C++来编写，但本文可以采用WebGL，以体现目前技术发展的趋势。程序大概分为以下几个部分：

1. 场景初始化

```
var scene = new THREE.Scene();
var allImages=new THREE.Group();
var camera = new THREE.PerspectiveCamera( 75,window.innerWidth/win
dow.innerHeight, 0.1, 1000000 );
var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

2. 数据加载

```
//在相机坐标系下创建相机坐标轴以及相机拍摄的图片
function imageMesh(fileName,w,h,f)
{
    var group=new THREE.Group();
    var loader = new THREE.TextureLoader();
    loader.load("../../images/temple/"+ fileName, function(texture
){
        var planeGeometry=new THREE.PlaneBufferGeometry(w/f*0.2,h/
f*0.2);
        var planeMaterial=new
THREE.MeshBasicMaterial({map:texture,side: THREE.DoubleSide});
        var planeMesh=new THREE.Mesh(planeGeometry,planeMaterial);
        planeMesh.translateOnAxis(new THREE.Vector3(0,0,1),0.03);
        group.add(planeMesh);
        var axis = new THREE.AxisHelper( 0.05 );
        group.add(axis);
    } );
    return group;
}

$.get("../../images/temple/temple_par.txt").success(function(conte
nt){
    //解析相机参数
    var lines=content.split("\n");
    var imageCount=parseInt(lines[0]);
    for(var i=0;i<imageCount;i++)
    {
        var line=lines[i+1].split(" ");
        var name=line[0];
        //K
        var k=$M([
                        [parseFloat(line[1]),parseFloat(line[2]),p
arseFloat(line[3])],
                                    [parseFloat(line[4]),parseFloa
t(line[5]),parseFloat(line[6])],
                        [parseFloat(line[8]),parseFloat(line[7]),p
arseFloat(line[9])],
                    ]);
        //R
        var r=$M([
                        [parseFloat(line[10]),parseFloat(line[11])
,parseFloat(line[12])],
                        [parseFloat(line[13]),parseFloat(line[14])
,parseFloat(line[15])],
                        [parseFloat(line[16]),parseFloat(line[17])
,parseFloat(line[18])],
                    ]);
        //t
        var t=$M([
                        [parseFloat(line[19])],
                        [parseFloat(line[20])],
                        [parseFloat(line[21])],
                    ]);
        //相机在世界坐标系中的位置
```

```
        var c=r.transpose().x(-1).x(t);
        var rt=r.transpose();
        var mesh=imageMesh(name,k.elements[0][2]*2,k.elements[1][2
]*2,k.elements[0][0]);
        //相机坐标系转换为世界坐标系
        var h=new THREE.Matrix4();
        h.set(rt.elements[0][0],rt.elements[0][1],rt.elements[0][2
],c.elements[0][0],rt.elements[1][0],rt.elements[1][1],rt.elements
[1][2],c.elements[1][0],rt.elements[2][0],rt.elements[2][1],rt.ele
ments[2][2],c.elements[2][0],0,0,0,1);
        mesh.applyMatrix(h);
        allImages.add(mesh);
    }
});
```

3. 渲染场景

```
scene.add(allImages);
camera.position.z = 2;
camera.lookAt(new THREE.Vector3( 0, 0, 0 ));
render();
function render() {
    requestAnimationFrame( render );
    renderer.render(scene, camera);
}
```

4. 鼠标事件（控制视点变换）

```
var clickX ; //鼠标点击X
var clickY ; //鼠标点击Y
var dragging = false; // 是否拖拽鼠标
var lastX = -1, lastY = -1; // 鼠标最后的位置
//添加鼠标事件
document.addEventListener( 'mousemove', onmousemove, false );
document.addEventListener( 'mouseup', onmouseup, false );
document.addEventListener( 'mousemove', onmousemove, false );
document.addEventListener( 'mousewheel', onmousewheel, false );
document.addEventListener( 'mousedown', onmousedown, false );
document.addEventListener( 'mouseout', onmouseout, false );
//以下是鼠标事件用于控制相机移动
function onmousewheel(event) {
    if(event.wheelDelta!=undefined)
    {
        camera.position.z-=event.wheelDelta / 20;
    }else
    {
        camera.position.z+=event.detail / 30;
    }
```

```
}
function onmousedown(ev){
    // 按下鼠标
    var x = ev.clientX, y = ev.clientY;
    // 鼠标在canvas范围内进行拖拽
    var rect = ev.target.getBoundingClientRect();
    if (rect.left <= x && x < rect.right && rect.top <= y&& y < re
ct.bottom) {
        // 记录鼠标位置
        lastX = x;
        lastY = y;
        // 开始拖拽
        dragging = true;
    }
}
function onmouseup(ev){
    // 鼠标弹起canvas拖拽结束
    dragging = false;
};
function onmouseout(ev){
    // 鼠标移出canvas拖拽结束
    dragging = false;
};
function onmousemove(ev){
    // 鼠标移动
    if (dragging) {
        // 只有鼠标拖拽时才计算
        var x = ev.clientX, y = ev.clientY;
        var dx = x - lastX;
        var dy = y - lastY;
        if (ev.altKey == 1) {
            // 鼠标点下同时按下alt键，x或者y方向平移
            var factor=0.5/window.innerHeight ;  // 移动比例
            dx = dx * factor;
            dy = dy * factor;
            camera.position.x-= dx;
            camera.position.y+=  dy;
        } else {
            // 鼠标点下，x或者y方向旋转
            var factor=100/window.innerHeight ; // 旋转比例
            dx = dx * factor;
            dy = dy * factor;
            allImages.rotation.x += dy * 0.01;
            allImages.rotation.y += dx * 0.01;
        }
    }
        lastX = x, lastY = y;
};
```

本程序还可以添加触屏事件，实现在手机上用手指控制视点位置。

总结

本文对明德学院提供的Temple多视图数据集进行了可视化。透过可视化可以直观的感受图片的拍摄过程。多视图三维重建的目标也就是用这样的多张视图重建出物体的三维模型。在写作本文的过程中，鄙人犯了一个小错误，用公式$(???)$时没有对R求逆（旋转矩阵的逆就是转置），导致调试花了不少时间，不服老不行啊。本文代码放在coding.net(https://git.coding.net/liujiboy/Furukawa2006.git)