

分类号 P22
U D C 528

密 级 公开
编 号 10486

武汉大学

硕士专业学位论文

室内 360 度全景浏览场景无缝过渡技术 研究

研 究 生 姓 名: 唐志雄

学 号: 2015282130087

指导教师姓名、职称: 胡庆武 教授

专 业 类 别 (领 域): 测绘工程

二〇一七年五月

A Study on Seamless Panorama Transition

Candidate: TANG ZHIXIONG

Student Number: 2015282130087

Supervisor: PROF. HU QINGWU

Major: Geomatics Engineering



School of Remote Sensing and Information Engineering

WUHAN UNIVERSITY

May, 2017

论 文 原 创 性 声 明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的研究成果。除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

学位论文作者 (签名):

年 月 日

摘要

360 度全景漫游系统是一种以静态图像为基础的初级虚拟现实技术，因其导览性好、真实性强而应用广泛。随着它的流行，其中存在的全景站点切换过渡不自然的问题愈发突出，亟待解决。由于场景切换问题，全景漫游系统不能有效衔接各全景站点，致使用户在场景切换时难以把握空间位置的变化、产生方向迷失感，极大影响了漫游系统的浏览体验。针对这一问题，尤其是室内场景的漫游过渡，本文重点做了如下两方面研究工作：

1) 全景浏览场景与三维场景模型的加载和融合方法研究。通过引入场景的三维模型来扩展全景漫游系统，使得漫游系统不仅可浏览各全景漫游站点，还可以跳出全景球的限制，游览整个三维空间。在如何衔接全景漫游和模型浏览这两种异质浏览交互方式问题上，提出通过生成临时观察相机并内插其空间位姿的方式来做到无缝过渡切换的解决方案。

2) 全景浏览场景之间的平滑过渡方法研究。通过提取同名控制点对生成匹配的三角网来引导场景过渡，使得场景过渡自然逼真。与常规的跳变、渐变、拉伸、干扰和视差法相比，动态全景图和动态全景球方案不存在重叠的虚影，场景内物体缓缓向侧边移动，视图自然地拉伸放大，给人自然走动观察的感觉，达到了良好的自然过渡效果。

最后，通过设计与实现室内 360 度全景场景无缝漫游原型系统实践验证了上述研究工作的有效性。

关键词: 室内全景漫游，场景自然过渡，网格渐变，纹理渐变，全景图与模型的融合

ABSTRACT

Panoramic Wandering System is a primary Image-Based Rendering System, which is getting more and more popular nowdays because of its realistic browsing effects. But some typical problems like the inconsistency of site browsering exist here haunting the developers for a sultion, baffling the users making them confused and lost in the system. To address these problems, this paper does two main research work:

1) Integration of panorama viewing and model browsing. By introducing 3D models to panoramic wandering system, we can not only browse panorama images, but also jump out of the viewing spheres and experience the real 3D environment. To tackle the inconsistency of these two incompatible browsing, we use a temporary viewing camera to make the visual effect realistic by carefully interpolating its trajectory and pose during the transition.

2) Smooth transition among panorama views. By providing control point correspondences, we can guide the view morphing from one panorama to another in a natual way. Compared with typical solutions like blending, stretching, and parallax-effect methods, this solution provides a natual transition effect similar to what people see when walking in 3D environment from one panorama viewing site to another.

By prototyping the indoor panoramic wandering system based on Three.js, we proved the effectiveness of these above proposed methods.

Key words: natural transition effect, mesh morph, texture morph

目 录

摘要	I
ABSTRACT	II
1 绪论	1
1.1 室内全景漫游系统	1
1.2 全景漫游技术研究现状	1
1.3 问题解决思路及本文组织结构	3
2 360 度全景与全景漫游关键技术	6
2.1 全景图及其投影方式	6
2.1.1 球面投影	6
2.1.2 立方体投影	7
2.1.3 Fisheye 投影	8
2.1.4 柱面投影	8
2.1.5 Plate Carrée 投影	9
2.2 全景漫游系统中的数学基础	11
2.2.1 坐标空间与坐标变换	11
2.2.2 旋转矩阵与四元数	14
2.2.2.1 欧拉角	15
2.2.2.2 欧拉角表达下的万向节锁问题	16
2.2.2.3 四元数	17
2.2.3 纹理的 UV 贴图	18
2.2.4 重心坐标及其与笛卡尔坐标的相互转化	20
2.3 模型数据与全景数据的采集和处理	21
2.4 本章小结	23
3 全景图与场景模型的加载和融合方法	24
3.1 全景图与场景模型间存在的矛盾	24
3.2 三维场景模型作为参考框架	25
3.3 全景站点位于三维场景模型之中	26
3.4 三维场景模型与二维全景漫游的自然衔接	27

3.5 本章小结	30
4 全景浏览场景间的平滑切换过渡方案	31
4.1 全景浏览的常见过渡处理	31
4.2 第一种解决方案：动态全景图	33
4.2.1 理论说明	33
4.2.2 同名控制点的获得	34
4.2.3 三角化与过渡	36
4.2.4 效果与评价	36
4.3 第二种解决方案：动态全景球	39
4.3.1 理论与试验	39
4.3.2 效果与评价	44
4.4 效率测试	45
4.5 本章小结	46
5 室内 360 度全景场景无缝漫游原型系统设计与实现	47
5.1 软件环境	47
5.1.1 基于 Three.js 的单页应用	47
5.1.2 用 Makefile 管理工程	49
5.2 实验数据	49
5.3 实现的功能点	51
5.4 工程上的一些考虑	53
5.4.1 全景图的压缩	53
5.4.2 全景图的分层切片后确保代码的可复用性	54
5.5 本章小结	55
6 总结与展望	56
6.1 内容总结	56
6.2 论文创新点	57
6.3 工作展望	57
参考文献	59
致谢	62

1 絮论

1.1 室内全景漫游系统

360 度全景漫游系统是一种以静态图像为基础 (IBR, Image-Based Rendering) 来虚拟现实场景的初级虚拟现实技术，具有对播放设备硬件要求低，显示分辨率高，导览性好，交互性强，真实性强，数据采集处理容易且数据量小便于网络传输等优良特性^[21]。

全景漫游系统的流行主要是因为它克服了真三维虚拟现实技术三个未能解决的问题^[13]: i) 三维数据获取难；ii) 数据量大网络传输难；iii) 对硬件要求高，显示效果和交互性性价比低。在数据获取上，室内全景漫游系统只需要使用数码相机配合鱼眼镜头在欲漫游地点采集多张影像数据，再使用 Pano Tools、PTGui 等工具即可拼接得到全景图。没有鱼眼镜头的情况下，也可用普通的数码相机多采集一些影像数据，并不需要特殊的采集设备。在网络传输上，生成的全景图用 JPEG 标准压缩后，一般 8192×4096 分辨率的全景图可以压缩至不足 2 MB，已达到了极好的体验效果；如果利用影像金字塔把分辨率降到 2048×1024 ，数据量不足 200 KB，而一般视角下的查看显示效果并没有明显损失；相比而言，三维模型数据动辄上百兆字节^[5]，三维细节还缺失严重，通过降采样压缩到 10 MB 以下的数据，往往只有轮廓，无法提供有效的三维漫游体验。在硬件设备上，无论是 PC 还是移动设备所配套的浏览器，均能很好地支持全景漫游系统。甚至在禁用 JavaScript 的浏览器上，通过 CSS 的三维变换或者 Canvas 的绘图功能也能提供全景漫游体验。而一般的虚拟现实技术对设备的图形处理能力要求很高，还需要安装专有软件才能使用。

1.2 全景漫游技术研究现状

1995 年，McMillan 和 Bishop 提出了基于图像的 360 度全景漫游方案，对采集到的原始影像数据进行一系列处理拼接得到全景图，在没有三维模型数据的情况下，即可提供有效的漫游效果^[13, 21]。在有三维模型的情况下，全景图还可以作为漫游的底图，弥补模型分辨率低、不够逼真等缺点^[36]。也有不少研究着眼于通过图像修补 (Image

Inpainting)、自动化地蒙版提取 (Automatic Matte Extraction)、光流等方法实现三维模型和全景图的融合和过渡。2006 年，华盛顿大学的 Noah Snavely、Steve Seitz 以及微软研究院的 Richard Szeliski，推出了 Photo Tourism 系统。只要导入一组大量的无序的影像数据，即可计算出每幅影像的位姿，构建出三维场景。通过系统自带的浏览界面用户可以漫游这些三维场景和二维影像，还可无缝地自如地切换，体验三维漫游^[29]。基于 Photo Tourism，微软于 2008 年正式发布 Photosynth，允许用户上传图片生成自己的场景模型。Photosynth 的全景切换效果逼真，极具美感^[11,28]。从此，基于全景的 360 度漫游系统开始在互联网流行^[8]。

由于真实三维信息的缺失，相比一般的虚拟现实系统 360 度全景漫游系统也存在一些问题。主要体现在：i) 漫游者的观察只能局限在特定的几个漫游点，不能在场景中任意移动，甚至当场景中只有一个视点时，只能实现虚拟场景的环视漫游；ii) 漫游点之间场景过渡时会发生跳跃式切换，效果并不自然，严重影响漫游体验。

对于前一个问题，可以增加漫游点采集数目，也可以通过画中游的方式模拟小幅移动摄像中心，以产生漫游点发生移动的感觉^[17]。但这种移动的幅度十分有限。对于后一个问题，一般有两种解决方法。第一种基于光场信息，通过采集大量的数据，创建全光函数，从而可以计算出任意视点的视图。第二种基于图像变形技术，在两个全景视图间进行一些图像处理^[23]，通常有四种策略：

1) 色彩混合 (Blend)

将两个全景浏览漫游视图沿着漫游方向对准后，进行色彩上的线性插值。这种方法可以提示用户场景正在从当前场景向下一场景切换，但是切换时的视觉效果无法反映出切换过程中观察点的空间位置变化，且无法处理场景中的遮挡问题。

2) 淡出和裁剪 (Fade and Cut)

这种方法只要求标记场景即可实现。全景图上某一特定区域的一个标签称为一个标记 (Tag)，在视点切换的时候，可以调整标记的区域，让把当前全景图的标记区域淡出，或者裁剪掉，然后加载新的全景图或者三维模型。最简单的标记法是直接把全景视图中的可视域作为标记区域，在标记区域加载新的全景视图。稍微复杂的标记方法是对当前视图和下一视图同一物体进行标注，对应的区域在淡出淡入时要进行图像变形。采用这种方案，如果有三维模型数据，无需特别考虑前后视图和模型的空间关系，只需要先淡出当前视图，再淡入模型，最后淡入新的视图遮住模型即可。

3) 内插过渡图像

这种解决方案从图像变换和插值的角度考虑，通过拉伸视图插入过渡图像来减弱切换过程的不连续对漫游体验的负面影响。但内插何种过渡图像以及如何获取这样的过渡图像又是新的难以解决的问题。

4) 视差法 (Parallax)

除了考虑现实物理规律，还可以参考人眼感知特性来提高过渡效果，这就是视差法^[30]。视差法分为全视差 (Full Parallax)、画中游 (Tour into the Picture)^[17] 和假视差 (Fake Parallax)。在全景过渡时采用全视差法，需要不断更新三维模型的视场角 (FOV, Field of View)，直到它消失于全景视图中。对视差建模是为了让场景过渡时自然地淡入、淡出三维模型。^[41] 指出，为了得到逼真的视觉效果，需要计算出每个模型到相机中心的距离，以及模型间的空间关系 (因为它们可能存在遮挡关系)，他们提出的 LAMP 表示法为全景视图提供了深度信息，而且与分层深度图像 (LDI, Layer Depth Image) 不同的是，LAMP 拥有真正的三维信息，相比全景视图它的视点是不固定的。考虑人眼对全景过渡中背景和前景有不同的心理预期，画中游将背景和前景分别处理，以期达到更好的过渡效果。^[17] 的方法则是先从全景图中提取粗略的背景图，再扣取遮挡物的蒙版并估计遮挡物蒙版下的背景颜色，最后在场景过渡时分别融合前后背景图和蒙版，使得过渡更自然。假视差法只是提取出背景的粗略几何信息，以及背景的颜色图层，通过简单的平移和缩放来模拟视差效果。实现较为容易。

可以看到，在 360 度全景漫游研究领域，尤其是解决场景过渡不自然这一课题上，研究趋势可总结为三点：

- 1) 用前后全景视图特征提取和匹配后再进行颜色混合取代直接渐变整个视图；
- 2) 除了物理上的合理性，人眼的视觉感知特点也被纳入了考虑之中，诞生了一系列通过视差法来提高过渡效果的解决方案；
- 3) 从准三维向真三维过渡，通过将三维模型数据引入基于全景图的 360 度全景漫游系统来改善全景视图切换过程。

1.3 问题解决思路及本文组织结构

如图 1.1，在解决全景漫游站点空间关系被破坏导致用户在全景漫游时无法感知位置变化而产生空间迷失感这一问题上，可从整体和局部两个角度考虑。从整体上，引

入模型数据作为整体参考框架统领全部全景漫游站点，可将问题转化为 1) 模型与全景在空间上的统一问题；以及 2) 模型浏览与全景浏览这两种不同的交互、浏览方式在时间上的衔接问题。在局部上，需要解决全景漫游场景间的平滑过渡问题，从纹理和模型两个角度考虑，前者可通过生成渐变的纹理贴图来使过渡自然，后者可通过移动全景球节点来变形全景场景以达到自然过渡的效果。因室外场景相邻漫游站点间不存在视觉上的关联性，无法做到视图上的自然过渡，所以本文只考虑解决室内简单环境近距离漫游切换下的场景自然过渡问题的解决。

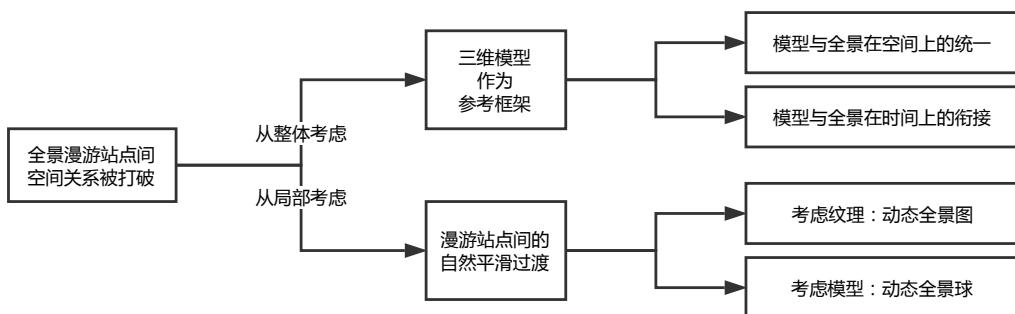


图 1.1 全景浏览场景无缝过渡问题的解决思路

根据上述解决思路，本文首先介绍室内全景漫游系统相关理论知识。先介绍几种常见的全景图投影，分析对比它们的优劣势，并解释了 Plate Carrée 投影流行的根本原因：正反向投影关系简单且数据采集、处理容易。接着介绍了漫游系统相关的多个坐标空间、纹理空间，以及三维物体的姿态表示法，串联了三维物体从模型到屏幕像素的变换过程、多种纹理贴图的基本原理和意义以及四元数在姿态表达上的特有优势。然后介绍了全景影像数据的采集和拼接相关的知识，并以 RIEGL RZ-400 激光扫描仪为例梳理了点云模型数据的采集和预处理流程。

介绍了相关理论知识后，引出全景漫游系统中普遍存在的全景站点间空间关联被打破以致漫游体验差的问题。先提出引入三维模型作为整体参考框架、全景站点置于其中的思路，通过内插临时观察相机位姿的方式来衔接两种观察交互方式的全景—模型过渡方案。又提出无缝过渡全景浏览场景的动态全景图和动态全景纹理方案，基于用同名匹配点对指导图形变形的原理，使得原本跳跃的场景切换变得如真实漫步于三维场景般自然，有效避免了空间迷失的感觉。

最后，介绍 360 全景浏览场景无缝过渡原型系统的设计原则、软件平台以及技术

路线。并根据实验数据缺乏三维模型的现状提出一种从全景图提取包围盒模型来代替真三维扫描模型的方法。还分享了原型系统开发中在全景图压缩，全景图分层、切片与加载解耦等相关问题上的经验。

全文的组织框架如下：

- 1) 第一章引出研究的课题：360 度全景浏览场景无缝过渡问题；
- 2) 第二章简介 360 度全景与全景漫游关键技术，包括全景图及投影方式、相关数学基础、全景和模型数据的采集和处理；
- 3) 第三章讲全景图与场景模型的加载和融合方法，主要从空间上的统一与时间上的衔接两个角度入手；
- 4) 第四章提出动态全景图和动态全景球两种全景浏览场景间的平滑切换过渡解决方案，分别从纹理和模型两个角度引导全景场景自然变形以达到类似步行漫游观察的效果；
- 5) 第五章介绍室内 360 度全景场景无缝漫游原型系统设计与实现，包含从理论到代码实践的全过程，验证了本文提出的全景图与场景模型的加载和融合方法以及全景浏览场景间的平滑切换过渡解决方案的有效性；
- 6) 第六章，总结与展望，总结在 360 度室内全景漫游场景过渡问题上做的一些思考、完成的工作以及创新点，并介绍下一步可探索的方向以及一些可能的思路。

2 360 度全景与全景漫游关键技术

本章简要介绍 360 度全景漫游系统相关理论知识，包括全景图及投影方式、相关数学基础以及全景和模型数据的采集和处理。

2.1 全景图及其投影方式

全景图是全景漫游系统的主要数据源，通过反投影将它映射到三维空间中，模拟三维环境来提供虚拟漫游体验^[20]。不同的应用场景和采集设备下，全景漫游系统使用的全景图可能采用不同的投影方式。依据全景图投影方式的差异，可将全景漫游系统分为几类：i) 球面投影漫游；ii) 立方体投影漫游；iii) 柱面投影漫游；和 iv) Plate Carrée 投影漫游。下面逐一介绍这几种的投影方式，并作简单比较。

2.1.1 球面投影

在几何学里，球极平面投影 (Stereographic Projection) 是一种将圆球面投影至平面的映射。在构造地质学里，它被称为球面立体投影或球面投影。除了投影点以外，投影在整个球面都有定义。在定义域内，球面投影具有光滑性、双射性和共形性 (保角性)，但不能维持距离、面积不变^[12]。

如图 2.1，球面球座标投影到平面极座标变换公式为：

$$(R, \Theta) = \left(\cot \frac{\varphi}{2}, \theta \right) \quad (2.1)$$

反向投影公式为：

$$(\varphi, \theta) = \left(2 \arctan \left(\frac{1}{R} \right), \Theta \right) \quad (2.2)$$

其中 φ 和 θ 是球面上点的天顶距和水平角， R 和 Θ 是球面点按照球面投影在南极切平面上的投影点的极坐标值。特别地，当 $R = 0$ 时， $\varphi = \pi$ 是球面和投影面的切点。易知，投影时若只投影一部分球面，畸变就会更小。图 2.2(a) 的 Panono 相机就是通过增加相机数目，减低每次球面投影的范围来保证形变量小。其表面有 36 个摄像头，拍摄得到的全景图达到 $16k \times 8k$ 分辨率。

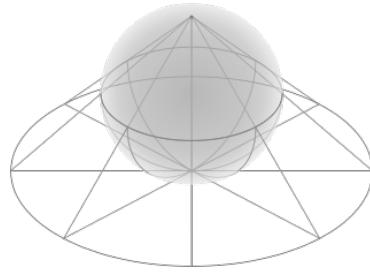


图 2.1 从北极点投影到南极切平面的球面投影

2.1.2 立方体投影

立方体投影是一种把部分或者整个球面投影到六个特殊平面的投影方式。这六个特殊平面可以围成一个立方体，立方体投影投影到每一个立方体平面时，采用的是 Rectilinear Projection (Gnomonic Projection)，视场角为 90°，变形较小^[34]。从中心浏览立方体即可实现基于立方体投影的全景漫游。

得到立方体全景图的方法一般有 3 种：i) 利用图形软件从其他影像合成无缝拼接在一个立方体六个面上的贴图；ii) 利用特定的拍摄仪器装置，分别在水平、垂直方向相隔 90° 的六个方向拍摄一张照片，再无缝拼接在一个立方体六个面上（如图 2.2(b)）^[44]；iii) 利用三维建模软件直接渲染合成。



(a) 球形全景相机 Panono (b) 相机架 + 六个 GoPro 相机 (c) 昂贵的 Matterport 相机

图 2.2 各种全景相机

在图形学中作 Cubemap 纹理贴图，立方体投影是一种较为理想的投影方式，因为纹理影像通常是计算机渲染而成的虚拟影像。而在全景漫游的应用中，使用立方体投影的全景图采集（需要获取上、下、左、右、前、后六个方向的影像）和使用（每个站点都需要加载六个方向的影像）都更麻烦。通常都只能从 Plate Carrée 投影得到的

影像间接生成立方体投影的纹理贴图。图 2.2(b) 采用了一个相机架子加上六个 GoPro 相机组来得到六个方向的影像，是一种常见的直接采集立方体投影全景图的方式。

2.1.3 Fisheye 投影

鱼眼 (Fisheye) 镜头为一种超广角的特殊镜头的俗称，因其前镜片球面直径短且呈抛物状向前凸出，类似鱼眼而得名。如图 2.3(a)，鱼眼镜头焦距短、视角广 (接近 180°)、景深大，一方面视域广远远超过人眼看见的范围，一方面成像中存在较大畸变，相片边缘处分辨能力差。



图 2.3 鱼眼镜头和全景相机

鱼眼投影的镜头相对简单，所以不少移动端全景采集设备都采用鱼眼投影。如图 2.3(b)，通过它的两个鱼眼摄像头输出的画面，各自涵盖了 180° 的水平和垂直视场角，然后将两个输出结果“扣”在一起就是全视域的沉浸式包围盒了。由于鱼眼相机视场边缘的分辨率较低，又因通过两个鱼眼镜头采集、拼接全景图时拼接线必会出现在视场明显区域，使用这种投影的漫游系统时浏览者很容易察觉到影像中的瑕疵。

2.1.4 柱面投影

柱面投影是指平面图像与柱面表面相互映射的过程，包括将平面图像投影到柱面表面的柱面正投影和将柱面表面的某个特定的观察区域投影到柱面的切平面上的柱面反投影。柱面投影是当前全景系统中应用最为广泛的一种投影方式，是柱面全景生成和显示过程中的必要环节。

如图 2.4(a)，设圆柱面的半径 $ON = r$ ，有： $\alpha = \arctan(\frac{x}{r})$ 。由于弧长等于半径乘

以弧度, 得 $x' = r\alpha = r \arctan(\frac{y}{r})$ 。如图 2.4(b), ΔOPP_1 相似于 $\Delta OP'P'_1$, $OP'_1 = r$, 这里的 OP_1 等于图 2.4(a) 中 $OP = \frac{r}{\cos \alpha}$, 故 $\frac{y'}{y} = \frac{r}{OP}$, 得 $y' = y \frac{r}{\cos \alpha} = y \cos \alpha$ 。

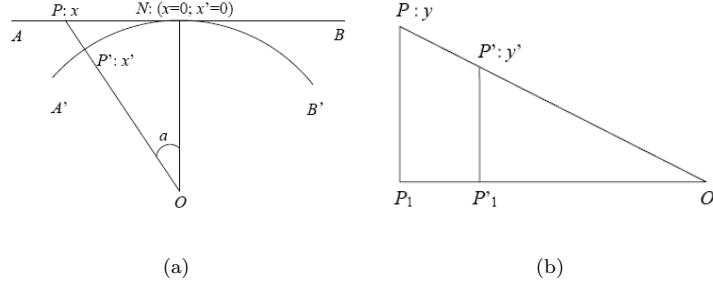


图 2.4

综合上述推导, 柱面上的坐标 (x', y') 与平面上的坐标 (x, y) 的正投影关系为

$$\begin{aligned}\alpha &= \arctan\left(\frac{y}{r}\right) \\ x' &= r\alpha \\ y' &= y \cos \alpha\end{aligned}\tag{2.3}$$

由上述正投影公式, 亦可推导出反投影公式:

$$\begin{aligned}\alpha &= \arctan\left(\frac{x}{r}\right) \\ x &= r \tan \frac{x'}{r} \\ y &= \frac{y'}{\cos \alpha}\end{aligned}\tag{2.4}$$

综上, 设 $P(x, y)$ 是相机拍摄的原始图像上任意一个像素, $Q(x', y')$ 是像素 $P(x, y)$ 投影到柱面全景影像的坐标。其中 W 、 H 和 f 为原始影像的宽和高以及相机焦距 (等于圆柱半径 r), 有

$$\begin{aligned}x' &= f \cdot \tan\left(\frac{x - \frac{w}{2}}{f}\right) + f \cdot \tan\left(\frac{W}{2f}\right) \\ y' &= \frac{f \cdot (y - \frac{H}{2})}{\sqrt{(x - \frac{w}{2})^2 + f^2}} + \frac{H}{2}\end{aligned}\tag{2.5}$$

2.1.5 Plate Carrée 投影

以上几种全景漫游技术中, 立方体投影多在虚拟环境中被用于 Cubemap 贴图以及 Skybox 贴图, Fisheye 投影多用于低成本全景影像采集, 基于柱面投影的全景漫游技术则濒于淘汰, 究其原因在于常规柱面投影无法实现全视域展示浏览。

Equirectangular 是一种特殊的柱面投影：等距圆柱投影。首先定义一些变量：i) λ : 欲投影点在球面上的经度；ii) λ_0 : 中央经线；iii) φ : 欲投影点在球面上的纬度；iv) φ_1 : 标准水平线，其上无距离畸变；v) x : 投影点在平面上的水平方向坐标；vi) y : 投影点在平面上的竖直方向坐标。则 Equirectangular 投影的前向投影可以表达为：

$$\begin{aligned} x &= (\lambda - \lambda_0) \cos \varphi_1 \\ y &= (\varphi - \varphi_1) \end{aligned} \quad (2.6)$$

反向投影可以表达为：

$$\begin{aligned} \lambda &= \frac{x}{\cos \varphi_1} + \lambda_0 \\ \varphi &= y + \varphi_1 \end{aligned} \quad (2.7)$$

当 φ_1 为 0, 这样的 Equirectangular 投影被称为 Plate Carrée 投影^[18,32]。如图 2.5(a), Plate Carrée 投影最常见应用是地球仪贴图。全景漫游系统与此类似, 只不过全景浏览时视线由球心向外而使用地球仪时视线由外向内指向球心。Plate Carrée 投影的特点是水平方向的坐标轴和经度一致, 水平视角的图像尺寸可以得到很好的保持, 垂直方向和纬度一致因而在垂直视角上尤其是接近两极的地方会发生无限的尺寸拉伸。

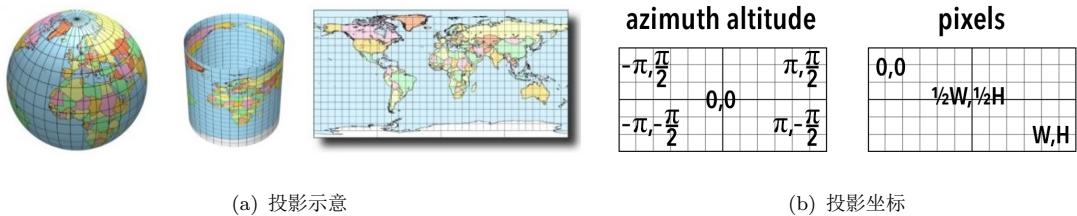


图 2.5 Plate Carrée 投影

全景漫游系统中全景图采用 Plate Carrée 投影有如下优势：i) 数据获取容易，可以从鱼眼相机拍摄的相片中拼接而成，如图 2.3(c)，在水平方向放置更多相机，而在天顶方向放置一个相机，即可拼接出优质的全景图，而且分辨率较低的上下部分远离全景漫游时的视线中心，不易被察觉；ii) 全景图宽高比固定为 2:1，正向和反向投影都简单，对全景图进行分层、切片以及加载都更容易。这些都促成了 Plate Carrée 投影成为全景漫游系统中全景图使用的最流行的投影方式。很多情况下，无论拍摄设备采用何种投影得到原始影像，都会被转化为 Plate Carrée 投影方式并存储成宽高比 2:1 的全景图。

2.2 全景漫游系统中的数学基础

理解和实现全景漫游系统需要掌握一些数学和图形学知识。这里简单介绍漫游系统中的坐标空间和坐标变换、旋转矩阵和四元数，纹理贴图等知识。

2.2.1 坐标空间与坐标变换

向量的表达和运算是线性空间的基础。在物理学中，向量被定义为方向和距离，空间中的一个起点和一个终点可以确定一个向量，平移一个向量，向量不变；在数学中，向量被抽象为满足数乘和向量加法的一种数；在计算机中，向量是定义了的一串数字。向量的表达取决于基底的选择，比如二维线性空间中，选取 $\vec{e}_1 = [1, 0]^T$ 、 $\vec{e}_2 = [0, 1]^T$ 作为基底，则任意向量可以表达为 \vec{e}_1 和 \vec{e}_2 的线性组合，即 $\vec{v} = x \times \vec{e}_1 + y \times \vec{e}_2$ 。这样，我们把 \vec{v} 参数化为 (x, y) ，表达成一个坐标。可见，向量可以根据基向量参数化为一串数字，用坐标来表达。这和空间中点的表达方式一致，所有也可以理解成固定向量起点用向量终点这个点来表示向量。

向量的集合被称为线性空间。当基底确定后，空间中每一个向量都有唯一的参数化表达。在不同的向量空间中，同一向量有不同的参数化结果。将一个向量从一个线性空间的参数化表达转化为另一个线性空间参数化表达的过程被称为线性变换，并可将一个线性变换参数化为一个矩阵。比如一个逆时针旋转 θ 角度的旋转变换可以参数化为 $R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$ 。

在图形学中，模型的每个顶点都需经过从模型空间不断变换到屏幕坐标的过程，依次经过的空间变换有：i) 模型空间 \xrightarrow{M} 世界空间；ii) 世界空间 \xrightarrow{V} 观察空间；iii) 观察空间 \xrightarrow{P} 裁剪空间；iv) 裁剪空间 \rightarrow 屏幕空间。

引入多个线性空间不仅便于参数化表达，更是因为一些概念只有在特定的坐标空间下才有意义，才更容易理解。不同的线性空间参数化为不同的坐标空间，形成一个层次结构，每个坐标空间都是另一个坐标空间的子空间。反过来，每个坐标空间都有一个父坐标空间。对坐标空间的变换实际上是在父空间和子空间之间对点和矢量进行变换。比如有父空间 P 以及一个子空间 C ，满足 $v_p = M_{cp}v_c$ 、 $v_c = M_{pc}v_p$ ，其中， M_{cp} 表示子坐标空间变换到父坐标空间的变换矩阵； M_{pc} 是反向变换，即父坐标空间变换到子坐标空间的变换矩阵，是 M_{cp} 的逆矩阵 M_{cp}^{-1} 。

下面简要介绍一下各个参数空间。

模型空间 (Model Space), 有时候也是对象空间 (Object Space) 和局部空间 (Local Space), 一般为右手坐标系, 自然方向为向右为 x , 向下为 y , 向前为 z 。一个简单三维模型比如球体, 一般以球心为模型空间中心, 将球面上所有三维节点参数化为模型空间中的向量 (即“点”), 即可表达这个三维模型。模型空间可以嵌套, 比如一张桌子可以简单分解成桌面和桌腿, 各自有自己的模型空间, 节点分别依照局部的模型空间参数化为一系列点位。子模型空间通过一个姿态矩阵将自己放置到父模型空间中, 通常用 4×4 矩阵 M_{local} 来表达。可以将此变换分解成 $M_{local} = M_{translation}M_{rotation}M_{scale}$, 看成常见的缩放、旋转和平移的组合。

有一个特殊的坐标系, 建立了我们所关心的最大的空间。将这个最外层的模型空间看成世界空间 (World Space)。通过模型变换 (Model Transform) 来将模型空间中的物体层层变换到世界空间, 这样, 每个模型空间都有一个世界坐标系下的姿态矩阵, 用 M_{world} 来表达。世界空间的选择取决于所研究问题的尺度和范围。一个全景漫游系统的世界坐标系可以选择全景球的自身的坐标系, 也可以选择为一个包含几张全景站点的房间, 也可以是一层楼, 一栋建筑。如果全景系统需要和地理信息系统联合, 世界坐标还可以选取常用的地理空间坐标系, 用经纬度来表达。

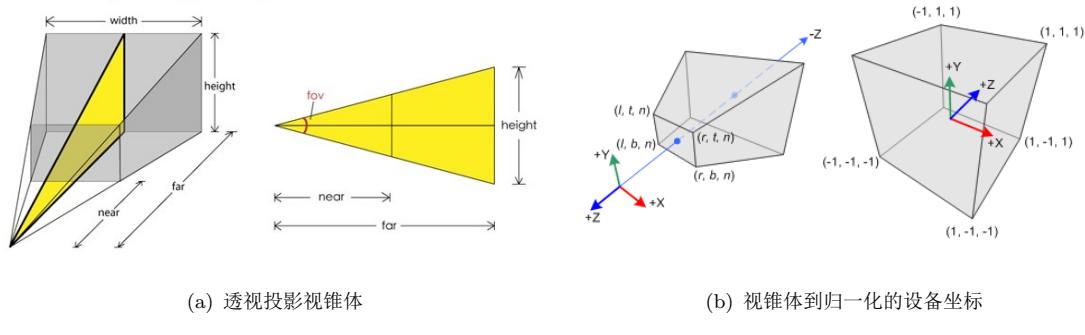


图 2.6 从观察空间到屏幕空间

观察空间 (View Space) 也被称为摄像机空间 (Camera Space), 是一个特殊的模型空间 – 摄像机的模型空间。裁剪空间 (Clip Space, 也被称为齐次裁剪空间) 用来去掉不需渲染的图元。裁剪矩阵 (Clip Matrix), 也被称为投影矩阵 (Projection Matrix), 可以用视椎体来表达裁剪的方案: i) 完全位于这块空间内: 保留; ii) 完全位于这块空间外: 剔除; iii) 与这块空间边界相交: 裁剪。通过透视投影和正交投影将渲染图元从观察空间投影到裁剪空间。如图 2.6(a), 观察空间到裁剪空间可用视椎体来表达。

透视投影满足：

$$\begin{aligned}
 nearClipPlaneHeight &= 2 \cdot near \cdot \tan \frac{fov}{2} \\
 farClipPlaneHeight &= 2 \cdot far \cdot \tan \frac{fov}{2} \\
 aspect &= \frac{width}{height} \\
 P_{clip} &= M_{frustum} P_{view} \\
 &= \begin{bmatrix} \cot \frac{fov}{2} & 0 & 0 & 0 \\ 0 & \cot \frac{fov}{2} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2 \cdot near \cdot far}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.8}
 \end{aligned}$$

经过投影后的坐标为 $X_{clip} = (x, y, z, w)$ 。如果 $-w \leq x \leq w$ 、 $-w \leq y \leq w$ 且 $-w \leq z \leq w$ ，图元保留。否则需要被剔除或者裁剪。如图 2.6(b)，裁剪矩阵改变了空间的旋向性 (从右手坐标系的观察空间到左手坐标系的屏幕空间)。正交投影的情况类似，这里不做赘述。

如图 2.6(b)，将裁剪空间的视锥体通过齐次除法 (Homogeneous Division) 转化为归一化的设备坐标 (NDC, Normalized Device Coordinates)，这就是屏幕空间。将裁剪空间转化到二维屏幕上完成节点的显示：

$$\begin{aligned}
 x' &= x/w \\
 y' &= y/w \\
 z' &= z/w
 \end{aligned} \tag{2.9}$$

经过齐次除法后，透视投影和正交投影的视锥体都变换到同一立方体内。OpenGL 中的这个立方体内空间满足 $-1 \leq x, y, z \leq 1$ ，将此空间内的点 (参数化为 $P_{clip} = (clip_x, clip_y, clip_z, clip_w)$) 转化成屏幕坐标 $p_{screen} = (screen_x, screen_y)$ ，即完成了节点从模型到屏幕的坐标变化过程。即：

$$\begin{aligned}
 screen_x &= \frac{clip_x \cdot pixelWidth}{2 \cdot clip_w} + \frac{pixelWidth}{2} \\
 screen_y &= \frac{clip_y \cdot pixelHeight}{2 \cdot clip_w} + \frac{pixelHeight}{2}
 \end{aligned} \tag{2.10}$$

其中 $pixelWidth$ 、 $pixelHeight$ 分别是屏幕宽度和高度 (单位：像素)。 z 分量 $clip_z$ 被用于深度缓冲，意为重绘屏幕时只有更靠近屏幕的点才可更新屏幕像素。

2.2.2 旋转矩阵与四元数

研究三维空间中一个物体的运动，应该给出物体上每一个点的运动轨迹，即将物体上每一个点的位置坐标表示成时间的函数 $X(t)$ 。但是对于刚体来说，只要指定一个旋转基点并建立局部坐标系便可以描述整个刚体的运动。这是因为刚体运动可以保证物体不变形，即其上任何两点之间的距离不随时间改变（如图 2.7(a)）。

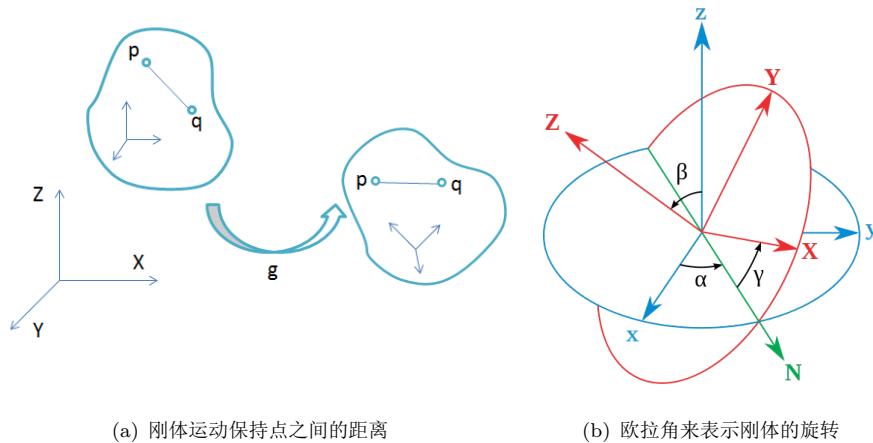


图 2.7 刚体的旋转运动及其欧拉角表示

旋转前后刚体上同一点前后的坐标转换可以看做是一个正交变换： $v' = Rv$ 。在三维欧氏空间中，这样的正交变换可用一个 3×3 的正交矩阵来表示，即

$$R = [\vec{e}_1, \vec{e}_2, \vec{e}_3] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad (2.11)$$

且 R 中元素满足六个独立的约束条件：

$$\vec{e}_i \cdot \vec{e}_j = \begin{cases} 0, & i \neq j, \\ 1, & i = j \end{cases}, \quad i, j \in \{1, 2, 3\} \quad (2.12)$$

所以 R 只有三个自由度，可以参数化为三个值来表示任意一个旋转。实际上应用中，常用欧拉角和四元数来表示旋转姿态，而不直接使用旋转矩阵。只有在进行坐标变换时，才将旋转姿态转化为矩阵进行运算。

2.2.2.1 欧拉角

欧拉角由 Leonhard Euler 引入，用来描述刚体在三维欧氏空间的姿态。刚体的旋转变换可以分解并看成绕着三个轴旋转，表示为 $(\theta_x, \theta_y, \theta_z)$ ，但变换顺序会影响旋转效果，因而需要指定旋转的次序。另一方面，旋转轴是旋转前的坐标轴，还是旋转后的坐标轴，也会影响旋转效果。旋转的顺序，以及每次旋角两轴的指定，并没有统一的规范，常用的组合就有 6 种，各个学科采取的默认标准也不同。所以在使用欧拉角来表达旋转时，应明确指定旋转顺序以及每次旋转的转轴。

如图 2.7(b)，定 xyz 坐标系是全局坐标系，XYZ 坐标系是局部坐标系。刚体旋转时前者保持不动，后者牢嵌于刚体随刚体一起旋转。记 xy 平面和 XY 平面的交点线为 N，ZXZ 顺序的欧拉角可参数化成三个数，为 (α, β, γ) ，其中 α 是 x/X 轴与 N 轴的夹角， β 是 z 轴与 Z 轴的夹角， γ 是 X 轴与 N 轴的夹角。具体旋转规则为：

- 1) 绕 Z/z 轴旋转 α 度，使 X 轴与 N 轴重合；
- 2) 绕 N 轴旋转 β 度，得到 Z 轴；
- 3) 绕 Z 轴旋转 γ 度，得到 X 轴。

可用旋转矩阵表示为：

$$\begin{aligned} \mathbf{R} &= \begin{bmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \beta & \sin \beta \\ 0 & -\sin \beta & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos \alpha \cos \gamma - \cos \beta \sin \alpha \sin \gamma & \sin \alpha \cos \gamma + \cos \beta \cos \alpha \sin \gamma & \sin \beta \sin \gamma \\ -\cos \alpha \sin \gamma - \cos \beta \sin \alpha \cos \gamma & -\sin \alpha \sin \gamma + \cos \beta \cos \alpha \cos \gamma & \sin \beta \cos \gamma \\ \sin \beta \sin \alpha & -\sin \beta \cos \alpha & \cos \beta \end{bmatrix} \quad (2.13) \end{aligned}$$

反向旋转用其逆矩阵 \mathbf{R}^{-1} 可表示为：

$$\mathbf{R}^{-1} = \begin{bmatrix} \cos \alpha \cos \gamma - \cos \beta \sin \alpha \sin \gamma & -\cos \alpha \sin \gamma - \cos \beta \sin \alpha \cos \gamma & \sin \beta \sin \alpha \\ \sin \alpha \cos \gamma + \cos \beta \cos \alpha \sin \gamma & -\sin \alpha \sin \gamma + \cos \beta \cos \alpha \cos \gamma & -\sin \beta \cos \alpha \\ \sin \beta \sin \gamma & \sin \beta \cos \gamma & \cos \beta \end{bmatrix} \quad (2.14)$$

可见，依照欧拉角的定义顺序可方便地将欧拉角表示的旋转变换转化为旋转矩阵表示，方便对向量做旋转变换。

2.2.2.2 欧拉角表达下的万向节锁问题

欧拉角表示旋转时，严格规定旋转顺序，且旋转顺序在外层的轴会影响内层轴，而内层轴不会影响外层轴。这样，当中间轴旋转 90° 与外层轴处于同一个平面时，旋转便失去了一个自由度，即出现万向节锁现象。

如图 2.8 所示，定坐标系旋转顺序为：Z 轴 \rightarrow Y 轴 \rightarrow X 轴。当绕 Y 轴旋转 90° ，再绕 X 轴旋转时会发现，实际上此时旋转的 YZ 平面与绕 Z 轴时旋转的 XY 平面是一个平面。此种情况下，旋转 X 轴和 Z 轴的效果一样，丧失了一个自由度。

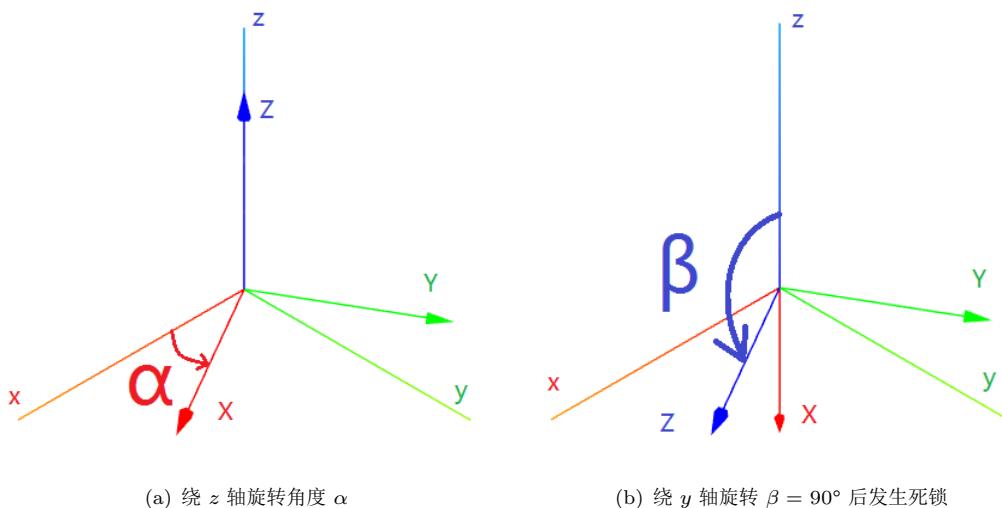


图 2.8 死锁现象

用旋转矩阵表示也可以发现这个问题：

$$\begin{aligned}
 R &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
 &\stackrel{\text{令 } \beta=90^\circ}{=} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.15) \\
 &= \begin{bmatrix} 0 & 0 & 1 \\ \sin(\alpha + \gamma) & \cos(\alpha + \gamma) & 0 \\ -\cos(\alpha + \gamma) & \sin(\alpha + \gamma) & 0 \end{bmatrix}
 \end{aligned}$$

2.2.2.3 四元数

四元数是哈密顿创造的超复数。四元数所在的空间 H 由实轴以及另外三个两两正交的轴 i, j, k 张成，其中 i, j, k 是特殊的复数单位，满足 $i^2 = j^2 = k^2 = ijk = -1$ 。将四元数定义为 $q = x\vec{i} + y\vec{j} + z\vec{k} + w = [x, y, z, w]$ ，参数化为 (x, y, z, w) 或者 $w + \vec{u}$ (其中 $\vec{u} = x\vec{i} + y\vec{j} + z\vec{k}$)。如果四元数可以表达为 $q = [x \cdot \sin \frac{\theta}{2}, y \cdot \sin \frac{\theta}{2}, z \cdot \sin \frac{\theta}{2}, \cos \frac{\theta}{2}]$ ，且满足 $x^2 + y^2 + z^2 = 1$ ，则称这样的四元数为单位四元数。

四元数满足如下定义的加法和乘法运算：

$$\begin{aligned} p + q &= (a + \vec{u}) + (t + \vec{v}) = (a + t) + (b + x)i + (c + y)j + (d + z)k \\ pq &= at - \vec{u} \cdot \vec{v} + a\vec{v} + t\vec{u} + \vec{u} \times \vec{v} \\ &= (bt + ax + dy - cz)i + (ct + ay + bz - dx)j + (dt + za + cx - by)k + \\ &\quad (at - bx - cy - dz) \end{aligned} \tag{2.16}$$

其乘法不可交换，特殊地，有：

$$\begin{aligned} ij &= k, \quad ji = -k \\ jk &= i, \quad kj = -i \\ ki &= j, \quad ik = -j \end{aligned} \tag{2.17}$$

从几何意义上，可以将 i, j, k 理解为三种旋转。其中 i 旋转代表 X 轴与 Y 轴相交平面中 X 轴正向向 Y 轴正向的旋转， j 旋转代表 Z 轴与 X 轴相交平面中 Z 轴正向向 X 轴正向的旋转， k 旋转代表 Y 轴与 Z 轴相交平面中 Y 轴正向向 Z 轴正向的旋转， $-i, -j, -k$ 分别代表 i, j, k 旋转的反向旋转。

常数分量 w 为 0 的四元数被称为纯四元数。存在于四维空间 H 的三维超平面上的一个向量 $p = (x, y, z, 0)$ 与三维欧氏空间中的三维向量 $p' = (x, y, z)$ 一一对应。这样我们就能用纯四元数来表示三维空间里的点^[40]。

又有 $R_q(p) = qpq^{-1}$ ，其中 q 为一个单位四元数，当输入 p 为纯四元数时，输出的四元数 $R_q(p)$ 也是一个纯四元数，这时可将 R_q 看成一个旋转，它将 p 对应的三维空间点旋转到了 $R_q(p)$ 对应的三维空间点处^[40]。这样一来，单位四元数 $q = [x \cdot \sin \frac{\alpha}{2}, y \cdot \sin \frac{\alpha}{2}, z \cdot \sin \frac{\alpha}{2}, \cos \frac{\alpha}{2}]$ 的几何意义为一个绕着轴 $\vec{u} = x\vec{i} + y\vec{j} + z\vec{k}$ 旋转 α 角的变换。相比欧拉角，它避免了万向节死锁问题。相比正交旋转矩阵，它的参数化

更紧凑，而且便于旋转插值，常见的插值运算 SLERP (球形插值) 就使用四元数来进行计算。如在球面上进行点的内插，便可使用四元数的内插，如图 2.9。内插公式为：

$$\begin{aligned}
 \text{Slerp}(q_0, q_1, t) &= q_0(q_0^{-1}q_1)^t \\
 &= q_1(q_1^{-1}q_0)^{1-t} \\
 &= (q_0q_1^{-1})^{1-t}q_1 \\
 &= (q_1q_0^{-1})^tq_0
 \end{aligned} \tag{2.18}$$

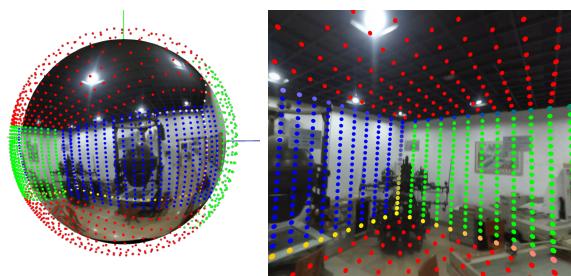


图 2.9 SLERP

2.2.3 纹理的 UV 贴图

一个三维物体包含两部分重要信息，其一是各节点位置，其二便是纹理。如图 2.10(b)，一个球体除了定义了各节点位置，还定义了一个 UV Map，记录了纹理贴图的方式，这样就能把地球表面图正确地贴到球体表面形成一个三维地球模型。OpenGL 支持一维、二维和三维纹理，其中最常用的是二维纹理，本小结只对二维纹理作简单介绍。

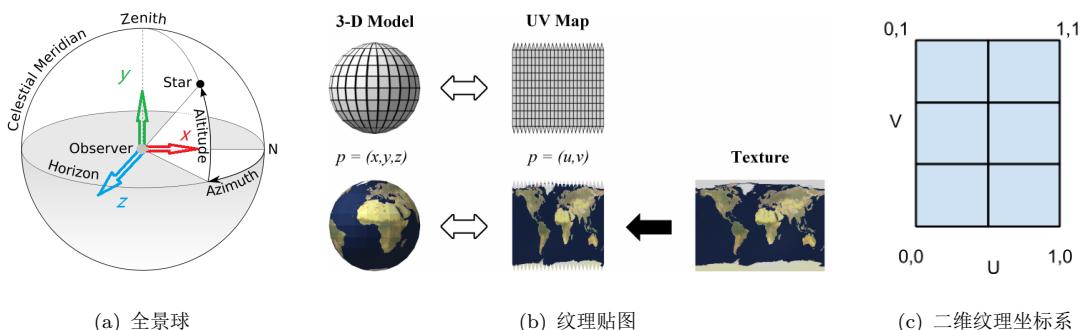
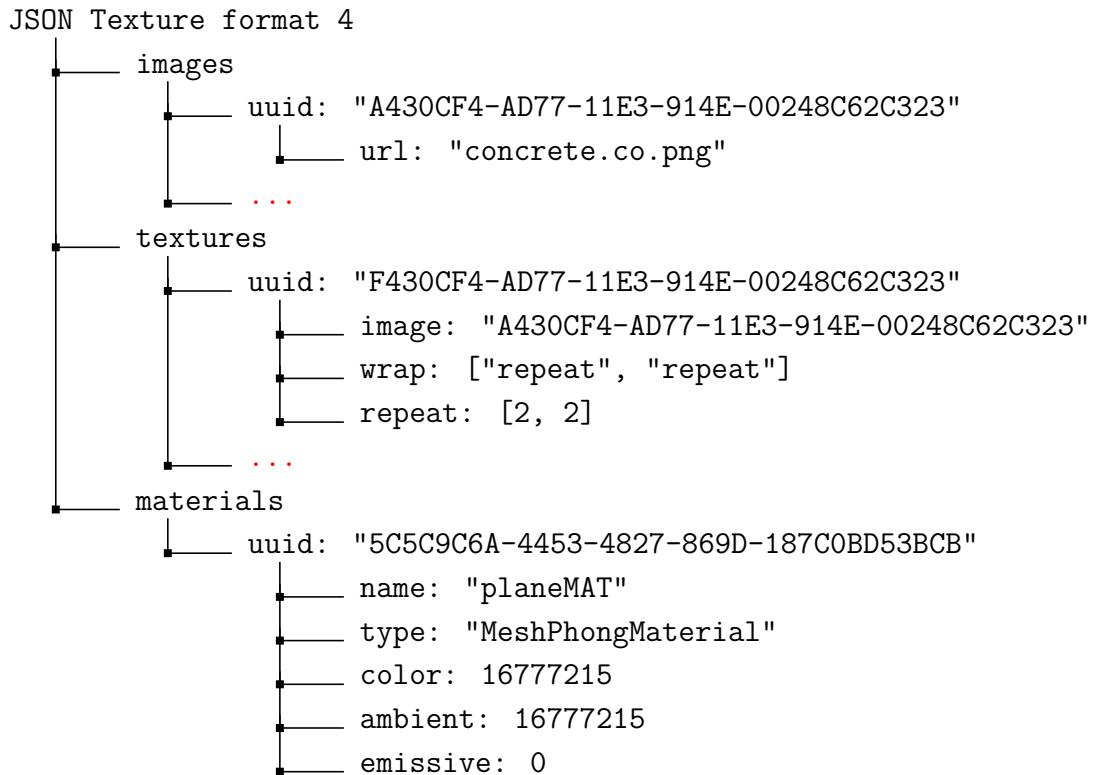


图 2.10 全景球纹理贴图

纹理贴图技术又称纹理映射技术，是计算机图形学中广泛应用的一项重要技术。传统的几何造型很难描述景物表面的微观细节，而利用纹理图像来描述景物表面各点处的反射属性可表现出物体表面丰富的纹理细节，提高计算机生成图形的真实性。采用纹理映射的方法可大大地简化建模的过程。纹理贴图除了常见的颜色纹理贴图，还可以是深度图 (Depth Map)，用来表达物体表面地凹凸起伏；也可以是法向图 (Normal Map)，用来表达物体反光时地法线方向，等等。在 Three.js 中，除了颜色纹理贴图，还有透明度纹理 (Alpha Map)、光照纹理 (Light Map)、发射度纹理 (Emissive Map)、镜面反射度纹理 (Specular Map) 等近十种不同用途的纹理贴图。

运用一个纹理贴图，就是把纹理图像根据纹理坐标对应到图元上。这就需要定义纹理图像坐标系。如图 2.10(c)，对于一个二维图像，将纹理坐标参数化为 (u, v) ，其中 u, v 的坐标都归一化到了 $[0, 1]$ ， u 从图像左侧到右侧对应 $0 \rightarrow 1$ ， v 从图像下侧到上侧对应 $0 \rightarrow 1$ 。比如，现在有一个三角形，顶点坐标分别是 V_1, V_2, V_3 ，以及一幅纹理图像。现在，我们可以在绘制三角形之前，为顶点 V_1, V_2, V_3 指定纹理坐标，即对每一个顶点配置一个 uv 坐标 $t_i = (u_i, v_i)$ ，其中 $i \in \{1, 2, 3\}$ 。使每一个顶点的坐标和它的纹理坐标一一对应。三角形内部的点，则会在 OpenGL 处理管线内被自动双线性插值^[19]。

Three.js 中一个纹理相关的定义大致如下：



```

    |
    +-- specular: 1118481
    |
    +-- shininess: 30
    |
    +-- side: 2
    |
    +-- opacity: 1
    |
    +-- transparent: false
    |
    +-- wireframe: false
    |
    +-- map: "F430CF4-AD77-11E3-914E-00248C62C323"

```

可以看到，一个纹理主要包含一个图片地址，以及纹理贴图的重复模式。当纹理被用到材料 (Material) 中时，可制定纹理的用途，比如上面提到的图像纹理、深度纹理、法向纹理。应用各种纹理，就能让三维物体具有相应的物理特性，产生更逼真的三维效果。

2.2.4 重心坐标及其与笛卡尔坐标的相互转化

绝对重心坐标系 (Absolute Barycentric Coordinates) 是一种特殊的参考坐标。如图 2.11，在三角形发生形变时，三角形内每个点在重心坐标系下坐标保持不变。这是十分重要且有用的一个性质。本小结简要介绍二维笛卡尔坐标和重心坐标的相互转化。

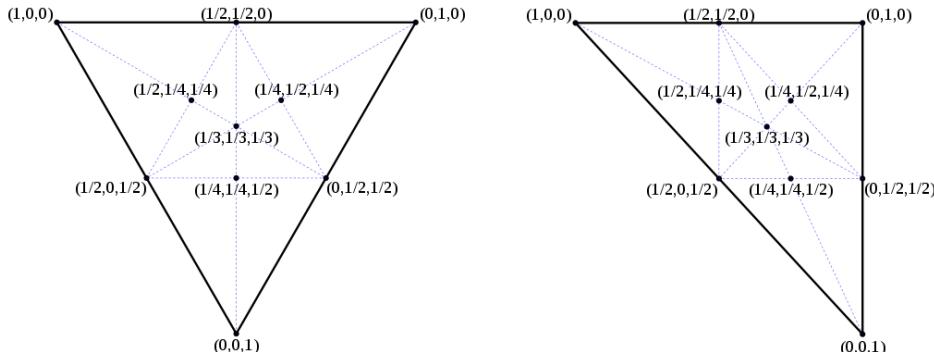


图 2.11 重心坐标系保持了点在三角形内的相对位置

通过三个顶点 A 、 B 和 C 来定义一个三角形 ΔABC 。设点 A 、 B 和 C 的笛卡尔坐标分别为 r_1 、 r_2 和 r_3 ，有 $r_i = (x_i, y_i)$, $i \in \{1, 2, 3\}$ 。则 ΔABC 内任一点 P 均可表示为三个顶点坐标的线性组合：

$$P = \lambda_1 \cdot r_1 + \lambda_2 \cdot r_2 + \lambda_3 \cdot r_3 \quad (2.19)$$

其中 $\lambda_1, \lambda_2, \lambda_3 \geq 0$ 且满足 $\lambda_1 + \lambda_2 + \lambda_3 = 1$ 。这样将 P 参数化为重心坐标系下的 $(\lambda_1, \lambda_2, \lambda_3)$ 。特别的， A 、 B 和 C 的重心坐标依次为 $(1, 0, 0)$ 、 $(0, 1, 0)$ 和 $(0, 0, 1)$ 。

则三角形内一个点 P 的笛卡尔坐标为 $r = (x, y)$, 重心坐标为 $\lambda = (\lambda_1, \lambda_2, \lambda_3)$, 满足:

$$\begin{aligned} x &= \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 = \lambda_1 x_1 + \lambda_2 x_2 + (1 - \lambda_1 - \lambda_2) x_3 \\ y &= \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3 = \lambda_1 y_1 + \lambda_2 y_2 + (1 - \lambda_1 - \lambda_2) y_3 \end{aligned} \quad (2.20)$$

这个式子可看成点 P 的重心坐标转化为笛卡尔坐标的转化公式。将它写成矩阵形式为 $\mathbf{T} \cdot \lambda = \mathbf{r} - \mathbf{r}_3$, 其中 $\mathbf{T} = \begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix}$, $\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$ 。这样, 就可以反算出 λ :

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \mathbf{T}^{-1}(\mathbf{r} - \mathbf{r}_3) = \begin{bmatrix} \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{\det(T)} \\ \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{\det(T)} \end{bmatrix} = \begin{bmatrix} \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)} \\ \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)} \end{bmatrix} \quad (2.21)$$

即完成了笛卡尔坐标向重心坐标的转化^[16, 35]。

2.3 模型数据与全景数据的采集和处理

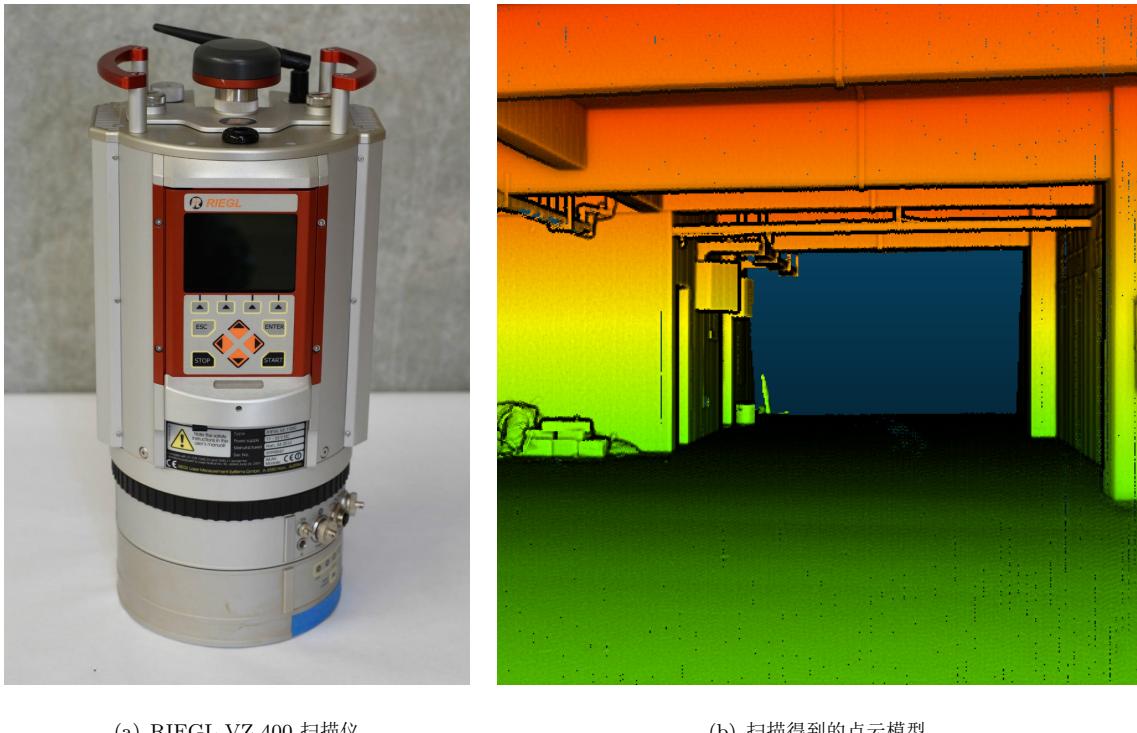
全景图拼接是全景图制作的关键环节, 弥补了普通相机视域无法覆盖 $360^\circ \times 180^\circ$ 大小的局限。图像拼接的重点在于能够精准地找出相邻两张图像中重叠部分, 确定它们的位置变换关系, 并进行拼接及边缘融合。

一般有两种方法采集全景影像:i) 使用特定的全景拍摄器材进行拍摄, 如图 2.3(b), 这种方法操作简便, 可直接导出全景图, 但拍摄设备价格昂贵, 难以普及; ii) 通过普通数码相机拍摄局部的图像, 再通过投影、拼接得到全景图, 此方法对拍摄要求较高, 需借助必要的相机支架等设备如图 2.2(b) 才能完成拍摄, 但成本低, 更灵活, 是当下主流全景图采集方案^[14, 20, 38]。

如图 2.3(c), 我们的全景采集设备固定在一个支架上, 水平方向配四个广角相机, 天顶方向配一个广角相机。采集的图片通过 PTGui 进行影像拼接得到 Plate Carrée 投影的宽高比为 2:1 的全景图。

Rigel (瑞格) 是一个有超过 30 年历史的激光扫描仪等测绘及地理信息相关产业的先进设备和测量系统解决方案的提供商。如图 2.12(a), 可使用 RIEGL VZ-400 远距离高速三维激光扫描仪来获得室内点云, 其测距能力为 600 米, 测量精度可选 3 毫米

或 5 毫米，有效测量速度为 122,000 点/秒，提供水平 360°、垂直 100° 的宽泛视场范围，而且采用了人眼安全的不可见 1 级安全激光。RIEGL VZ-400 得到的点云密集精准，如图 2.12(b)。



(a) RIEGL VZ-400 扫描仪

(b) 扫描得到的点云模型

图 2.12 RIEGL 激光点云扫描仪

但直接加载大量点云，对内存和显示渲染模块都是极大挑战，所以要对点云进行其使用 Point Cloud Library (PCL) 进行预处理，并作格式转换和导出。RIEGL VZ-400 可导出为 i) ASCII (*.*); ii) Wavefront (*.obj); iii) VRML (*.wr1); iv) PLY (*.ply); v) LAS (*.las); vi) KMZ (Google Earth) 等格式。其中最流行的三维格式是 Obj，最初是 Wavefront 的标准 3D 模型文件，定义了对象的几何和其它的一些特性。因其文件结构非常简单，极适合在应用程序中作三维模型读取或进行三维文件格式的转换，因此被广泛应用于各种应用软件中，比如 Maya、Blender^[27]。Obj 格式的三维模型既可保存成 ASICI 文本，后缀名为 .obj，程序和人眼都易于辨识；也可以是二进制文件，后缀名为 .mod，导出的模型文件体积更小。由于 OpenGL 中提供了最基本的有多边形构造三维模型的方法，因此可以方便地从三维图形数据文件中读取 Obj 模型数据并在 OpenGL 中绘制。我们将点云转化为面模型并导出为 Obj 格式。将三维模型和全景数据整合到一起需要考虑两者不同的坐标系统，下一章将介绍这部分内容。

2.4 本章小结

本章简要介绍了室内全景漫游系统相关的 i) 与全景图相关的常见投影如球面投影、立方体投影、Plate Carrée 投影，并分析了不同投影的优缺点及适用场合，理解全景图的投影有利于制定合适的全景数据采集方案，有利于理解全景处理流程以得到适当的全景图产品；ii) 数学基础，涉及了图形学中常见的多种坐标空间、三维物体姿态的三种表示法（旋转矩阵、欧拉角、四元数）及其优缺点以及二维纹理贴图相关知识，对坐标空间和对纹理贴图相关知识的了解分别是融合三维模型和全景数据和实现场景自然过渡的首要条件；iii) 模型数据和全景数据的采集和处理，介绍了全景采集设备以及影像拼接相关知识、RIEGL VZ-400 激光扫描仪及其点云处理、模型导出相关知识。

3 全景图与场景模型的加载和融合方法

通过引入三维场景模型来统领各全景浏览漫游站点可提高漫游系统的整体感。本章讲述如何在空间上统一场景模型与全景浏览球以及如何在时间上衔接与此相关的两种不同交互方式。

3.1 全景图与场景模型间存在的矛盾

在虚拟三维空间中，全景图通过全景球实现全景的漫游浏览。在空间上统一全景球与三维模型，需要考虑它们之间的位置关系以及比例的设定。如果全景球太大，在三维空间中与模型一起查看时会显得很不协调；如果全景球太小，难以在模型浏览时进入全景站点的浏览。如果两者位置没有放置正确，会误导用户，影响其对空间位置的把握。这些全景图和场景模型存在显著的区别，导致两者的加载和融合不是简单将两者载入到三维环境中，需要考虑如何将两个坐标空间转换过渡到统一的世界坐标中这一问题。

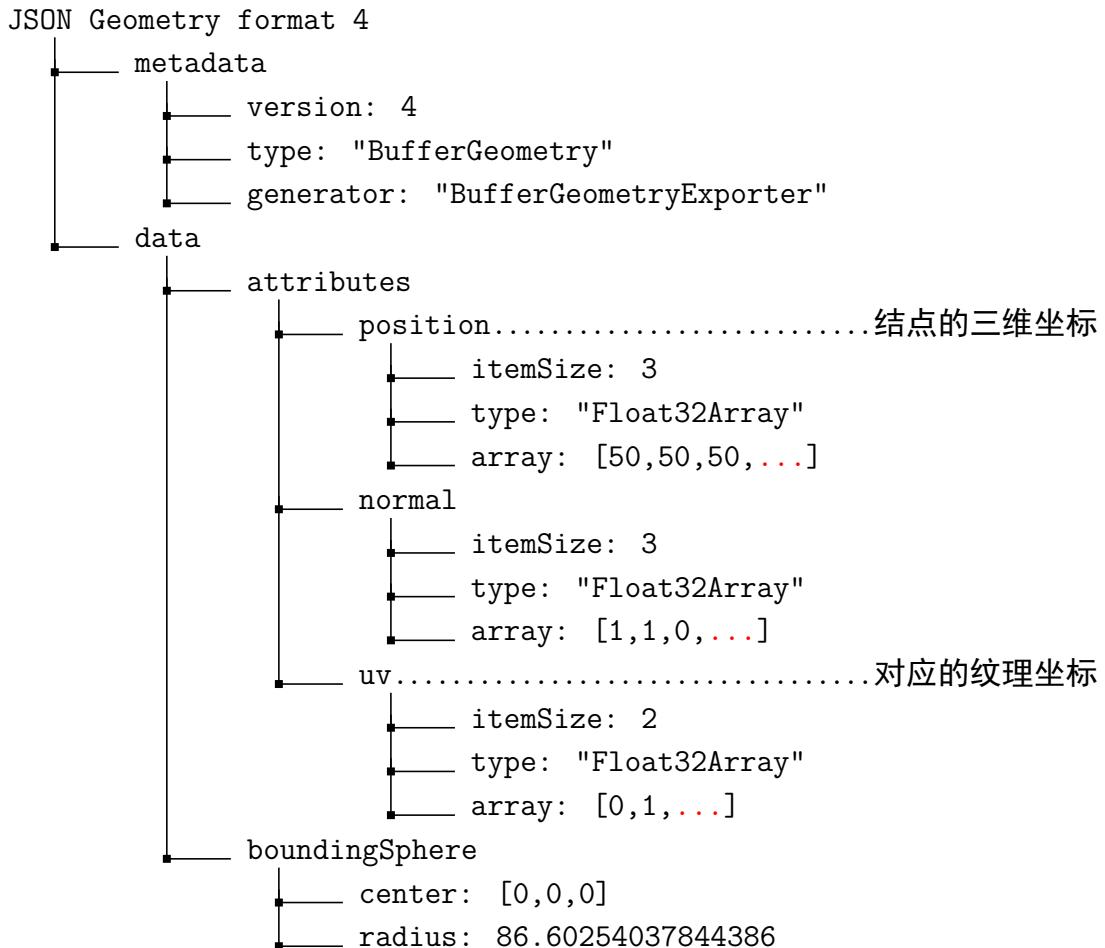
另外，从观察方式上两者也存在明显差异。对于全景球而言，将观察相机放置在全景球球心来浏览全景场景，通过改变观察相机的姿态来调整观察视角以观察全景场景的当前浏览部分。此时，全景球是固定的，观察相机的位置是固定的，唯一改变的是观察相机的姿态，以及最终观察到的内容。而对于三维模型的浏览观察，观察相机的位置是移动的。OrbitControls 是一种常见的三维模型交互控件，可以代为调整相机的观察姿态。通过它来操作观察相机时，观察相机的空间位置不断在三维球体中调整，并保持相机视线中心指向模型，此时模型固定而观察相机不固定。这些观察方式上的区别，导致全景图与场景模型在加载和融合时需要处理观察相机的位置和姿态调整问题以及交互方式的切换问题。

可以看到，全景图与场景模型间存在时间与空间上的矛盾。可通过如下三个步骤化解矛盾：

- 1) 三维场景模型作为参考框架；
- 2) 全景站点位于三维场景模型之中；
- 3) 三维场景模型与二维全景漫游的自然衔接。

3.2 三维场景模型作为参考框架

室内全景漫游系统若无需和其他系统联动，其主参考框架 (Reference Frame) 即世界坐标系便可直接选定为漫游场所这栋楼或者几个房间。首先就需要在漫游系统中载入三维场景模型。不同模型数据的数据格式各不一样，但提供的信息基本一致。以下 Three.js 中模型数据为例：



可见，模型数据就是几何结构 (Geometry) 和材质 (Material) 的组合，前者记录了模型各节点的位置信息，后者记录了模型的纹理贴图。

因模型的保存格式不同，加载时要按照相应数据规范进行解析，才能在三维空间中将模型绘制出来。首先生成一个局部坐标系，加载模型数据中的点、线、面。对点而言，重要的是正确渲染点的大小和颜色；对线而言，则可能需要指定一维纹理，做色彩的过渡；对面而言，则需要指定纹理以及各节点的纹理坐标。最后，如果模型数据的坐标系统旋性和渲染系统所用的坐标系统旋性不同，还需要对其中某一个方向，比如 z 轴作反向处理。如果不是直接将模型作为世界坐标，还需要指定模型的位置和姿

态，如代码 3.1 所示。

代码 3.1 设定模型的位姿

```

1 var R = new Matrix4();
2 R.set(
3     rotation[0][0], rotation[1][0], rotation[2][0],      0,
4     rotation[0][1], rotation[1][1], rotation[2][1],      0,
5     rotation[0][2], rotation[1][2], rotation[2][2],      0,
6     0,           0,           0,           1 );
7 // 设置模型的位置和姿态
8 model.position.set( x, y, z );
9 model.quaternion.setFromRotationMatrix( R );

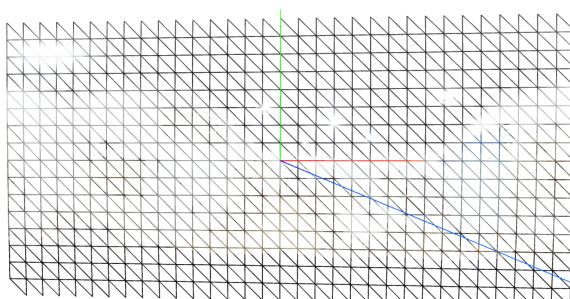
```

需要注意的是模型数据中旋转矩阵和渲染模块中使用的矩阵的存储方式可能不一致。一般而言，作文档和交流时人们使用行主序的矩阵，而作计算机内部存储时采用列主序。在载入姿态信息时需要留意是否需要提前做一次矩阵转置。

3.3 全景站点位于三维场景模型之中

一张全景图渲染在一个全景球上，便形成了一个可漫游的全景站点。只要记录了全景图采集时的位置和姿态，通过一定的坐标变换就能将全景球放置在模型这个参考框架下的正确位置上，位姿的设置依旧可参考代码 3.1。

全景球可以看成全景图贴合在球体上形成的观察球。如图 3.1(a) 和图 3.1(b)，在平面点上生成规则的一系列节点，将全景图附上，设定好纹理坐标。然后将节点移动到球面上，如图 3.2(a)、3.2(b)、3.2(c)，形成球体。最后贴上纹理即可进行全景站点的浏览，如图 3.2(d)。这里需要注意将平面转化为球面时应当保证全景图“贴在”内侧，否则观察效果有误。简单地，可通过反向 x 轴实现。



(a) 全景球节点坐标



(b) 全景球纹理坐标

图 3.1 全景球节点和纹理坐标

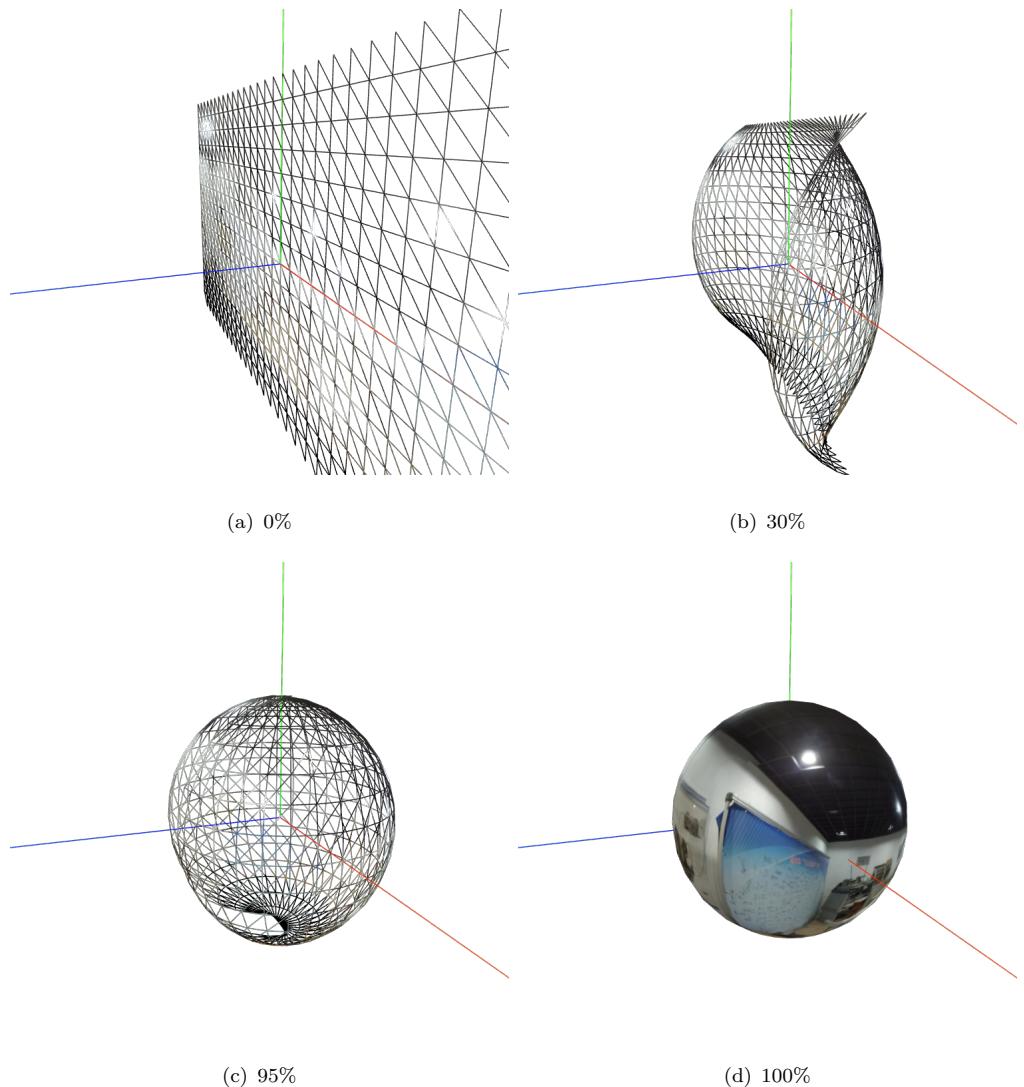


图 3.2 全景球纹理贴图

最后，将相机置于全景球中心并提供基本的相机姿态调整功能，即可模拟人在三维场景中定点环视漫游的效果。

3.4 三维场景模型与二维全景漫游的自然衔接

浏览一个三维模型，通常使用 OrbitControls 的方式来作交互，此时模型被放置在中心位置，通过移动观察相机切换视角来查看模型。浏览一个全景，通常将相机固定，通过改变相机自身的姿态来观察全景球（如图 3.1）的不同部位，实现全景的漫游。如果只使用一个观察相机，则需要许多变量来记录相机当前的状态，以便操纵相机从一种交互模式向另一种交互模式过渡，逻辑处理较为复杂。为了方便起见，可在三维环

境中保留了许多相机，通过在相机之间进行过渡来保证三维场景模型与二维全景漫游的自然衔接^[1]。

设全景站点 P 处于模型 M 中，分别通过相机 C_p 和 C_m 观察。图 3.3(a) 是作为全局参考框架的三维场景模型，为武汉大学遥感楼仪器展览馆的一间。 C_m 当前观察结果为图 3.3(b)，大致朝着 x 轴正方向，图中正中心的球体为全景站点 P 所在的全景球。 C_p 的观察结果为图 3.3(c)，大致朝向 x 轴负方向。当用户点击全景球后，视角应当从 C_m 的当前状态逐步过渡到 C_p 的当前状态，并在交互方式上作相应调整。

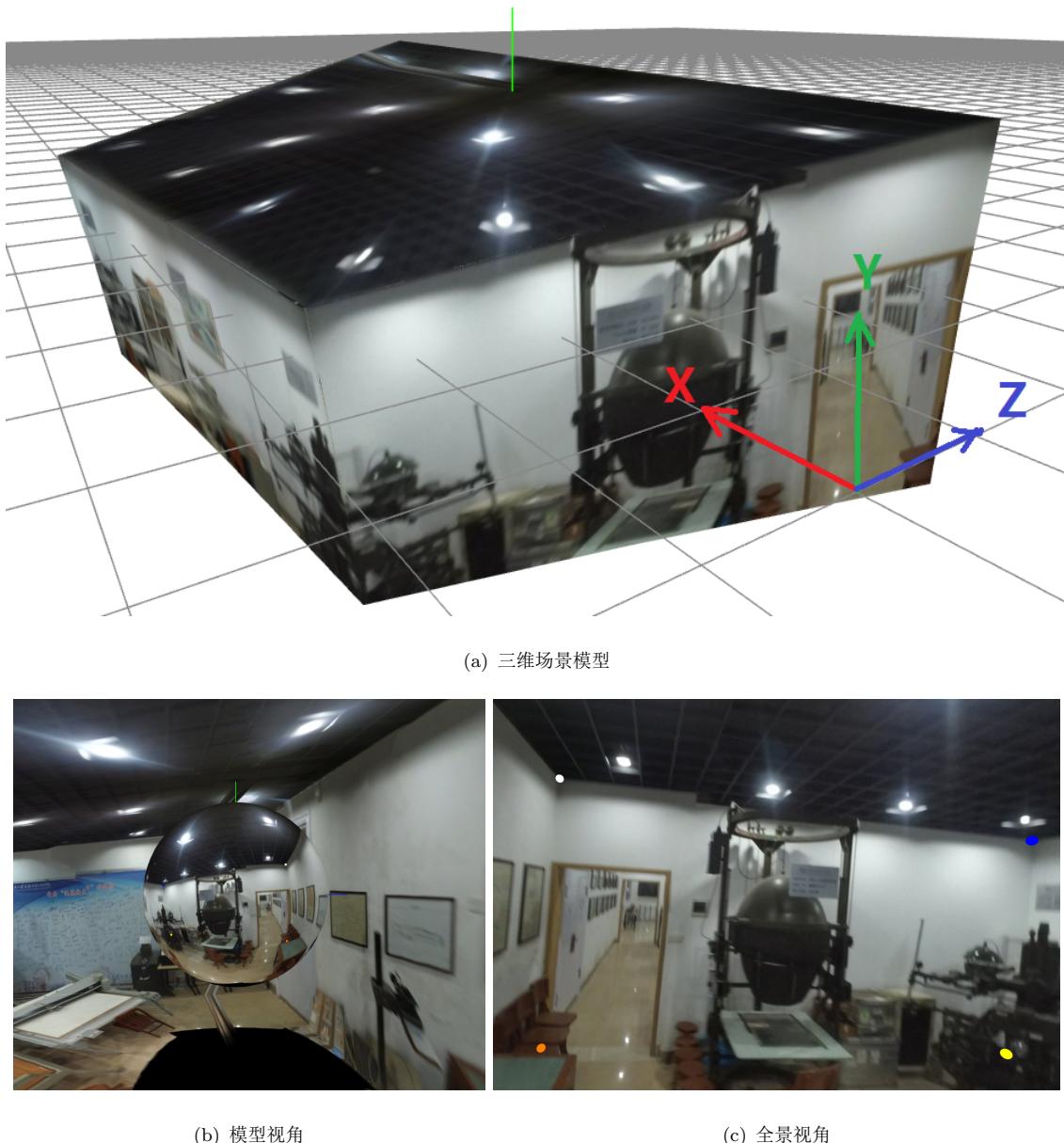


图 3.3 三维二维切换

为了让过渡更加自然，引入如下逻辑。首先，引入一个临时观察相机 C_x 用作过渡。在过渡期间，用户观察的结果均为 C_x 看到的内容。当用户开始从模型观察模式切换到全景观察模式时（或者反过来）， C_x 先从 C_m 获取姿态和相机配置（比如相机视场角、远近截平面等参数），这样一来， C_x 的观察结果将与 C_m 一致。同时，当启用 C_x 后，原本 C_m 的交互方式失效，此时不再响应用户的移动视角等操作，用户界面进入自动漫游状态。 C_x 从初始状态 C_m 向目标状态 C_p 的过渡，即可完成模型向全景自然过渡的过程。具体而言，运动轨迹参考 [6, 25, 39]，使用三次贝塞尔曲线来拟合出三维螺旋线（以 C_m 、 C_p 的当前位置和方向作为四个控制点）；姿态使用四元数在 C_m 和 C_p 两个姿态间内插（SLERP）得到每一时刻的相机朝向（见参考代码 3.2）。最终效果如图 3.4。

代码 3.2 过渡相机的位姿内插

```

1 // 算法：内插出时刻 t 时临时观察相机 Cx 的位姿
2 // 输入：全景观察相机 Cp，模型观察相机 Cm，时刻 t
3 // 输出：Cx 在 t 时的位姿
4
5 // 初始化三次贝塞尔曲线轨迹
6 var strengthM = 2,
7     strengthP = 1;
8 var trajectory = new CubicBezierCurve(
9     Cm.position.clone(),
10    Cm.position.clone().add(getTargetDirecton(Cm).multiplyScalar(strengthM)),
11    Cp.position.clone().add(getTargetDirecton(Cp).multiplyScalar(strengthM)),
12    Cp.position.clone()
13 );
14
15 // 参数化过渡过程中每一个时刻为 t (0 <= t <= 1)
16 // 渲染时根据 t 值更新相机的位姿
17 Cx.position.copy(
18     trajectory.getPoint( t )
19 );
20 Cx.quaternion.copy(
21     Cm.quaternion.clone().lerp(Cp.quaternion.clone(), t)
22 );
23
24 // 从相机获得朝向向量
25 function getTargetDirecton(cam) {
26     var dir = new Vector3(0,1,0);
27     var R = new Matrix4().makeRotationFromQuaternion( cam.quaternion );
28     return dir.applyMatrix4(R).normalize();
29 }
```



图 3.4 场景到全景的过渡

3.5 本章小结

本章介绍了室内全景漫游系统的中全景图和三维场景模型加载和融合相关事宜。从全景图浏览方式和模型浏览方式的差异和矛盾引出全景图和模型的场景融合问题：如何自然地从一个观察视角自然过渡到另一个观察视角，并作交互方式的调整。最终，通过在过渡期间应用线性插值和球形插值调整一个临时观察相机的姿态来衔接全景浏览漫游与场景模型查看，解决了全景图与场景模型加载与融合的问题。

4 全景浏览场景间的平滑切换过渡方案

全景过渡问题是全景漫游系统中一大关键点和难点。如果给予用户足够的视觉引导，过渡效果自然逼真，系统的真实性会更好，浏览体验效果也会更优异。如果不能很好的解决这个问题，用户在使用过程中便可能无法理解空间位置的变化，感到困惑不解。就此问题，本章提出动态全景图与动态全景球两种全景浏览场景间平滑切换过渡方案，前者通过提取同名控制点形成动态三角化格网来更新原有全景图的方式形成供全景过渡时用的动态全景图，在场景切换时应用动态全景，即可从当前站点向下一全景站点自然平滑过渡。后者通过同名控制点形成动态三角化格网覆盖在原全景球上，通过全景球自身的变形实现全景场景的平滑过渡。

4.1 全景浏览的常见过渡处理

常用的过渡处理可以简要总结为五类：跳变、渐变、拉伸、干扰和视差法。

在具体介绍这几种过渡处理前，首先定义几个概念：

全景站点 用户漫游全景图时固定不动的视线中心的位置，也是全景影像拍摄时相机的位置，也叫全景漫游点，简称站点；一个全景站点包含了与此全景图相关的位置、北方向等信息。

场景过渡 在一个全景站点浏览时，从某一个方向进入另一个全景站点的过程，本文关心的是用户在场景过渡这一过程的视觉体验；理想的场景过渡应让用户体验到亲身步行漫游于三维环境的感觉。

同名点 三维空间点在多张二维影像数据中成的像点，被称为一组同名点。

下面一一介绍这五种常见的全景漫游过渡处理。

1) 跳变

跳变指的是场景过渡时，直接从一个全景站点切换到另一个全景站点，从一张全景图切换到另一张全景图。严格地说，跳变不能算作一种场景过渡的解决方案，因为它直接无视了直接切换对浏览体验带来的负面效果，并未采取任何策略来削减这种不好的浏览体验。遗憾的是，很多室内全景漫游系统都通过跳变的方式来过渡全景站点。跳变使得用户点击场景切换的按钮后，无法理解过渡时的空间变化，产生了一种方向

迷失的感觉。这样的场景过渡割裂了室内各全景站点间的联系。

2) 演变

渐变是一种相对跳变而言的简单过渡处理，可提供基本的场景过渡感。如图 4.1(a)，如果把从全景站点 a 的显示跳变到全景站点 b 的显示表达为 $a \rightarrow b$ ，渐变则是 1) $a \rightsquigarrow b$ ，从全景站点 a 的显示逐步过渡到全景站点 b 的显示；或者 2) $a \rightsquigarrow x \rightsquigarrow b$ ，先将全景站点 a 的显示逐步过渡到 x 状态，再逐步过渡到全景站点 b 的显示。方法 1) 是在两个全景站点的显示效果中进行颜色融合，对每个像素，有 $C_{ij} = (1 - \alpha) \times C_{ij}^{(a)} + \alpha \times C_{ij}^{(b)}$ ，即对每个显示位置 (i, j) ，将当前全景站点的颜色和下一站全景的颜色进行因子为 α 的线性插值，得到一个折中的视觉效果。在场景过渡时，内插因子 α 从 0 逐步过渡到 1，即完成了从场景 a 到场景 b 的跳转。方法 2) 与方法 1) 类似，只不过在两者之间插入第三个状态，通常为全黑或者全白，即先淡出当前全景，再淡入下一站点的全景。



(a) 渐变

(b) 拉伸

(c) 干扰

图 4.1 一般全景过渡方法

3) 拉伸

拉伸是另一种过渡方案，基于一个简单的假设：从当前全景站点过渡到下一个全景过渡站点时，由于距离空间中各物体更近，在显示效果上，物体都会出现不同程度的放大。所以可以采用从中心向四周拉伸的方式，提供一种“进入”的感觉。如图 4.1(b)，简单的图像上的拉伸处理，可以给用户一种向前移动的感觉。但拉伸的处理更适合于室外全景漫游比如街景全景漫游。因为街景漫游时前进方向固定为道路方向，且空间中物体距离都相对更远，视野更开阔，拉伸处理的效果也较为不错。但在室内，由于出现较多的空间遮挡，移动的距离近，拉伸的处理只会给人一种夸大的空间位移感，并不自然逼真。

4) 干扰

如渐变处理的方法 2) 引入第三种状态来避免两种全景混合在一起产生凌乱感，干

扰也通过类似的策略来解决场景过渡难题。如图 4.1(c)，干扰法通过生成视觉上的一些“噪音”来分散用户的注意力，避免用户注意到场景过渡的不自然。常用的干扰就是在屏幕上产生像素块的跳动、扭曲，或者色调的变化。

5) 视差法

除考虑现实物理规律，还可参考人眼感知特性来提高过渡效果，这就是视差法。视差法分为全视差 (Full Parallax)、画中游 (Tour Into the Picture)^[17] 和假视差 (Fake Parallax)^[24]。全视差需要精细三维模型的支持，假视差法只是提取出背景的粗略几何信息，以及背景的颜色图层，通过简单的平移和缩放来模拟视差效果^[4, 36, 43]。视差法主要用于在全景站点附近作左右的移动来模拟人眼对着一个物体变换视角查看的过程，与场景过渡时从当前全景站点向下一全景站点移动的方向和尺度都有明显的差别。而且视差法需要真三维模型的支持，而一般全景漫游系统没有这些数据^[42]。也因数据获取上的问题，本文提出的解决方案无法和视差法的效果做对比测试。

4.2 第一种解决方案：动态全景图

如果可以生成一个渐变过渡的动态全景图，在全景过渡时“播放”这个动态全景，即可实现自然地场景过渡^[30]。这一小节提供了一个简单的思路来生成这样一个动态全景纹理。

4.2.1 理论说明

首先，输入为两张全景图（见图 4.2），目标是从第一个全景图自然过渡到第二个全景图。



(a) 全景图 a

(b) 全景图 b

图 4.2 输入影像

考虑全景站点 a 向全景站点 b 的过渡过程。两个全景站点的全景图分别为全景图 a (图 4.2(a)) 和全景图 b (图 4.2(b)), 记作 $I^{(a)}$ 和 $I^{(b)}$ 。在 $I^{(a)}$ 和 $I^{(b)}$ 对应区域 (比如在漫游前进方向左右各取 60° 形成两个矩形) 上找 N 对同名点 P_i , $i \in \{1..N\}$ 。同名点对 P_i 在 $I^{(a)}$ 上坐标为 $P_i^{(a)} = (x_i^{(a)}, y_i^{(a)})$, 在 $I^{(b)}$ 上坐标为 $P_i^{(b)} = (x_i^{(b)}, y_i^{(b)})$ 。 $\{P_i\}$ 在全景图 a 上形成一个点集 $\{P_i^{(a)}\}$ 。Delaunay 三角剖分算法具有空圆、最大化最小角特性, 利于保持网格结构 [3,9], 所以将点集 $\{P_i^{(a)}\}$ 用 Delaunay 三角剖分算法三角化为 M 个三角形 $T_j^{(a)}$, $j \in \{1..M\}$ 。

对每一个三角形 $T_j^{(a)} = \Delta ABC$, 它的三个顶点为 $A = P_{j1}^{(a)}$ 、 $B = P_{j2}^{(a)}$ 和 $C = P_{j3}^{(a)}$ 。 A 、 B 、 C 均来自点集 $\{P_i^{(a)}\}$ (即 $j1, j2, j3 \in \{1..N\}$), 因此可根据同名点对找到它们在 $I^{(b)}$ 上的对应点, 记作 $A' = P_{j1}^{(b)}$ 、 $B' = P_{j2}^{(b)}$ 和 $C' = P_{j3}^{(b)}$ 。 A' 、 B' 、 C' 形成三角形 $T_j^{(b)} = \Delta A'B'C'$ 。

对每个 $T_j^{(a)}$, 生成一个三角面片 $Face_j$, 其顶点初始化为 $P_1 = A$ 、 $P_2 = B$ 和 $P_3 = C$, 并添加 $I^{(a)}$ 和 $I^{(b)}$ 两个纹理贴图, 前者的透明度设置为 $TRANSPARENT(I_a) = 0$ (完全可见), 后者透明度设为 $TRANSPARENT(I_b) = 1$ (完全不可见)。三角面片 $Face_j$ 的三个顶点的六个 UV 纹理坐标为分别取自 A 、 B 和 C 在 $I^{(a)}$ 的纹理坐标, 以及 A' 、 B' 和 C' 在 $I^{(b)}$ 的纹理坐标。

设参数 α 为全景站点 a 向全景站点 b 场景过渡的进度。特殊地, 当 $\alpha = 0$, 显示全景漫游场景 a; 当 $\alpha = 1$, 显示全景漫游场景 b。随时间变化逐步调整 α , α 由 0 变 1 的过程, 即是全景漫游场景过渡的过程。则本过渡方案则可以描述为: 在场景过渡任意时刻, 三角面片 $Face_j$ 的三个顶点坐标分别取 $P_1 = LERP(A, A', \alpha)$ 、 $P_2 = LERP(B, B', \alpha)$ 和 $P_3 = LERP(C, C', \alpha)$, 其中的 $LERP$ 是线性插值函数, 满足 $LERP(f, t, \alpha) = (1 - \alpha) \times f + \alpha \times t$ 。同时, 三角面片上纹理的透明度分别满足 $TRANSPARENT(I_a) = \alpha$ 、 $TRANSPARENT(I_b) = 1 - \alpha$ 。三角面片覆盖在原纹理之上, 重新采样形成新的纹理帧。这些纹理帧作为整体就是一个可用于全景场景自然过渡的动态纹理。

4.2.2 同名控制点的获得

在本过渡方案下, 严格配准的同名点点对保证了全景过渡的自然。因此获取同名控制点点对是本方案重要的一环。

通过特征点提取算子、匹配算法来提取同名点点对。如图 4.3, 可以看到通过 SIFT 特征点算子虽然提取并匹配到不少特征点, 但这些点有很多问题: i) 提取到的特征点分布不均匀且偏离了场景前进的中心方向; 而且 ii) 这些点并不是人眼关注的内容, 比如在人眼不关注的门框内提取了很多特征点, 在门框上却只提取出了一个特征点。因为特征提取算子无法提取出优质的控制点, 只得采取人工方法来做这部分的工作。如图 4.4(a), 用定制的同名点编辑器打开两张全景图, 通过鼠标和键盘添加一些同名控制点点对 (如图 4.4(c))。在编辑器里, 两张全景图的同名点被连接起来。此编辑器提供基本的同名控制点加载保存、增删、编辑等操作。在使用它人工添加同名控制点点对时, 应当考虑用户浏览全景时的视觉焦点。比如这里, 就应当在门框的四个角添加控制点, 因为人眼很可能把门框作为参考框架。



图 4.3 特征点提取匹配 (SIFT 特征)

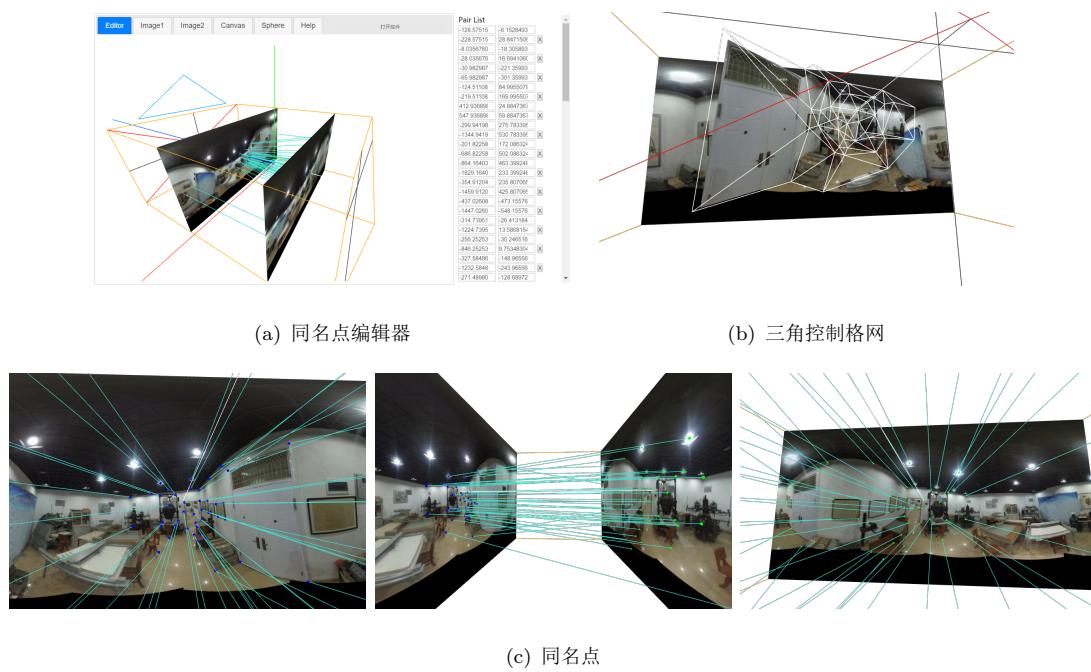


图 4.4 同名控制点的编辑和三角控制格网的生成

4.2.3 三角化与过渡

如图 4.4(b), 将同名控制点在 $I^{(a)}$ 上的点集 $\{P_i^{(a)}\}$ 三角化。现在讨论最简单的情况, 那就是一个三角面片 Face 如何过渡到另一个三角面片。如图 4.5(a), 当一个三角形 ΔABC 向位置 $\Delta A'B'C'$ 过渡时, 在 α 时刻, Face 的三个顶点坐标 P_1 、 P_2 和 P_3 分别是 A 和 A' 、 B 和 B' 、 C 和 C' 的线性插值。同时, 如图 4.5(b) 内部点的颜色过渡通过 $C_{ij}^{\alpha} = (1 - \alpha) \times C_{ij}^{(a)} + \alpha \times C_{ij}^{(b)}$ 内插获得, 其中 $C_{ij}^{(a)}$ 为 ΔABC 上重心坐标为 (i, j) 的像素值, $C_{ij}^{(b)}$ 为 $\Delta A'B'C'$ 上重心坐标为 (i, j) 的像素值。对 $\{T_j\}$ 中每个三角形生成一个三角面片, 并记录下每个 α 时刻的纹理, 记作 $I^{(a) \xrightarrow{\alpha} (b)}$, 这样就生成了从 $I^{(a)}$ 到 $I^{(b)}$ 一系列的纹理帧 $\{I^{(a) \xrightarrow{\alpha} (b)}\}$, 形成一个动态纹理 $I^{(a) \rightsquigarrow (b)}$ 。在全景球上“播放”动态纹理, 即可获得场景自然过渡的效果。

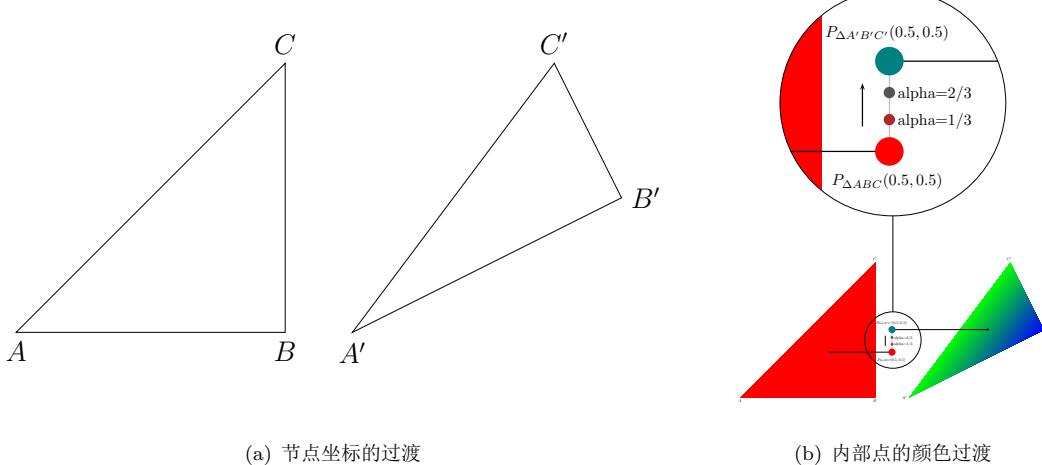


图 4.5 三角面片的过渡

4.2.4 效果与评价

使用上述方法采样生成 4096×2048 的动态纹理, 全景浏览的效果如表 4.1。第一列是 α 值, 第二列是同名控制点三角化效果, 当 α 值由 0 到 1 变化时, 三角形 $\{T_j^{(a)}\}$ 逐步向 $\{T_j^{(b)}\}$ 过渡。各三角面片覆盖在原始纹理图片之上, 更新下方的像素, 形成新的纹理。将这个新的纹理贴图到全景球上, 即为观察到的场景切换过程, 见表格第三列。作为参照, 第四列是普通的采用简单渐变的过渡效果。可以看到, 相比简单渐变, 动态纹理没有重叠的虚影, 无论是上方的灯、左侧墙上的画幅、地上的桌子还是右侧的仪器设备, 都缓缓向侧边移动, 视图自然地拉伸放大, 给人自然走动观察的感觉。可

以看出，采用动态纹理的方法确实可以达到自然过渡场景的效果。

但动态全景图方案存在潜在的性能问题。由于 α 变化后，需要重新渲染形成新的全景纹理，这一过程计算量（主要是点集的三角化，以及顶点位置的线性内插）不大，但现场渲染 Canvas 纹理十分消耗内存，而且应用更新了的 Canvas 纹理到全景球上，也有一定的性能负担。再者，渲染动态全景图需要单独的渲染循环，和全景查看的主要渲染循环存在资源竞争关系，所以动态全景图方案下，很可能出现不流畅乃至卡顿的现象；在移动端浏览器中，还可能因内存、显存不足，发生卡死现象。

总而言之，相对跳变、渐变、拉伸、干扰以及视差法，动态全景图方案能在仅有全景数据的情况下，实现极为自然的场景过渡切换，是一个有效的室内全景浏览场景无缝过渡解决方案。

表 4.1 动态全景图效果

α	动态全景图 (示意)	动态全景图 (效果)	简单渐变 (效果)
0.0			
0.1			
0.2			
0.3			

			
0.4			
0.5			
0.6			
0.7			
0.8			
0.9			
1.0			

4.3 第二种解决方案：动态全景球

4.3.1 理论与试验

利用同名控制点对形成匹配的三角面片，并根据同名点进行位置和颜色的线性过渡，以形成动态纹理的每一帧，再在全景球上应用这些纹理帧，即可实现自然的全景场景过渡。这是动态全景图方法的基本原理。动态全景图方案的最大问题在于需要不断重绘形成新的纹理并重新进行纹理贴图，这一过程内存占用较大，可能会导致过渡过程帧率下降乃至不流畅、卡顿等问题。动态全景球方案试图通过移动全景球相关节点的方式来避免重新生成纹理和重新贴图的过程。

借鉴动态纹理的方法生成同名控制点对的方法，生成 $\{P_i^{(a)}\}$ 和 $\{P_i^{(b)}\}$ 。借鉴动态纹理的方法生成三角网的方法，生成 $\{T_j^{(a)}\}$ 。对每一个三角形 $T_j^{(a)} \in \{T_j^{(a)}\}$ ，找到全景球上在此三角形内的节点（节点的定义参照图 3.1(a)），假设有 K 个，设为 $\{V_k^{(a,j)}\}$ ，其中 $k \in \{1..K\}$ 。对每个节点 $V_k^{(a,j)}$ ，求其在 $T_j^{(a)} = \Delta ABC$ 内重心坐标系下的坐标 $(\lambda_1^{(a,j,k)}, \lambda_2^{(a,j,k)})$ 。并找到 $I^{(b)}$ 上对应的三角形 $T_j^{(b)} = \Delta A'B'C'$ 相同重心坐标 $(\lambda_1^{(b,j,k)}, \lambda_2^{(b,j,k)}) = (\lambda_1^{(a,j,k)}, \lambda_2^{(a,j,k)})$ 点的笛卡尔坐标 $V_k^{(b,j)}$ 。这样，我们确定了全景球上一个节点 $V_k^{(a,j)}$ 的形变目标位置 $V_k^{(b,j)}$ 。在动态纹理中， ΔABC 的位置逐步过渡到 $\Delta A'B'C'$ ；在动态全景球中， $V_k^{(a,j)}$ 的位置逐步过渡到 $V_k^{(b,j)}$ 。

在全景球节点 $V_k^{(a,j)}$ 的几何位置逐步过渡到 $V_k^{(b,j)}$ 时，还需要对其纹理进行变形。首先为原全景球生成另一个纹理，这个纹理使用全景图 $I^{(b)}$ ，纹理坐标不变。这样，节点 $V_k^{(a,j)}$ 有两个纹理坐标 $(u^{(a,j,k)}, v^{(a,j,k)})$ 和 $(u^{(b,j,k)}, v^{(b,j,k)})$ 。根据 $V_k^{(a,j)}$ 的目标位置 $V_k^{(b,j)}$ 计算新的纹理坐标，并赋值到第二个纹理坐标，即 $(u^{(b,j,k)}, v^{(b,j,k)}) = lonlat2uv(V_k^{(b,j)})$ ，这里的 $lonlat2uv$ 将球面经纬坐标转化为纹理 UV 坐标。同样，在移动节点的同时，需要相应改变全景球两组纹理的透明度。

但实际测试后发现，这个动态全景球方案中存在一些不可调和的漏洞。如图 4.6(a)，其中的蓝色节点为初始位置，绿色节点为目标位置，黄色节点为当前位置，测试过程中，各个节点正确地从初始位置过渡到了目标位置，但出现了全景球面部分“翻折”的问题，继而导致颜色融合失效，效果很不自然。另外如门框处还发生了弯折扭曲，反映了这一方案不能有效保持常见三维物体的曼哈顿特性^[10]。其主要原因在于，如果直接操作全景球，部分节点被拉开后，势必导致面的“翻折”和“重叠”（因全景漫游站

点切换时视图基本呈放大趋势); 而且在选取同名控制点点对时就假设了图像变形时三角面片作为整体保持不变, 但反算全景球上节点形成的三角形和原来三角形并不重合, 通过这样的方法来操作视图不能保持真实空间中物体的结构特点。

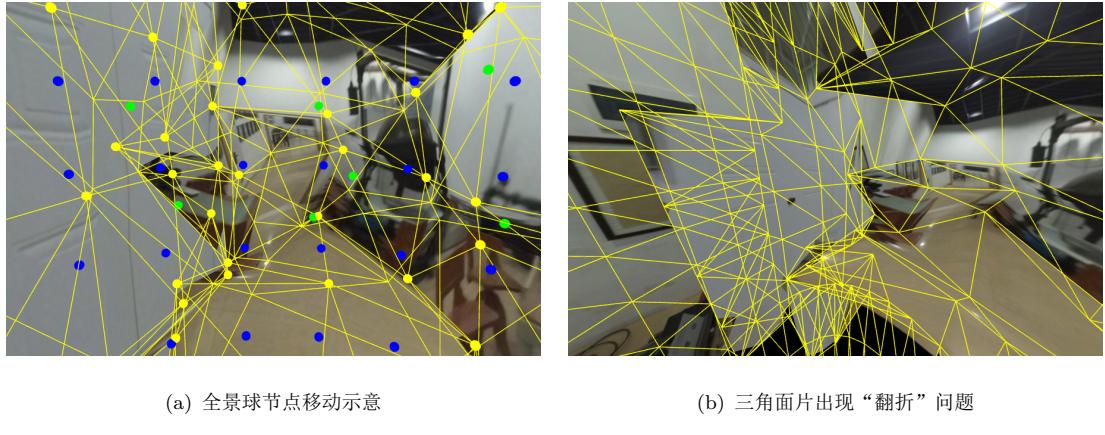


图 4.6 原始动态全景球方案中存在的问题

改进 1: 在三维空间中应用三角面片

动态全景球方案虽然存在较大问题, 但避免了全景图的实时渲染, 现针对上述问题改进这一方案。借鉴动态全景纹理生成的过程, 考虑在三维空间中应用三角面片: 对每一个三角形 $T_j^{(a)} \in \{T_j^{(a)}\}$, 生成三角面片 $Face_j$, 并根据三角形位置在上面应用 $I^{(a)}$ 和 $I^{(b)}$ 两个全景图作为纹理贴图。再根据同名控制点点对将三角面片 $Face_j$ 的位置由 $T_j^{(a)}$ 过渡到 $T_j^{(b)}$ 并相应地渐变两纹理的颜色内插比例来达到自然过渡效果。需要注意的是, 动态全景图方案中三角面片的位置过渡 (如图 4.5(a)) 采用 LERP (线性插值) 即可, 而动态全景方案中在三维空间中过渡三角面片的位置, 需要采用 SLERP (球形插值)。实际上, 经纬坐标系下的 LERP 等效于球面上的 SLERP。

可以看到, 动态全景图方案与动态全景球方案都使用同名控制点点对来作视觉引导, 都使用 Delaunay 三角剖分算法来生成三角面片, 都使用颜色线性插值来作纹理混合, 唯一不同的是动态全景图方案上在二维平面上应用三角面片变成了动态全景球方案下在三维球面上应用的三角面片, 并相应地用 LERP 和 SLERP 来插值处过渡时期三角面片各顶点的位置。

最终效果如表 4.2, 第二列为动态全景球示意, 可以看到三角面片的移动以及应用纹理后的效果; 第三列是动态全景球方案的过渡效果; 作为参照, 第四列是动态全景图方案在同一位置的过渡效果。

可以看到，动态全景图和动态全景球方案在 $\alpha \geq 0.2$ 后视觉效果几乎一致。只有在 $\alpha < 0.2$ 时，动态全景球方案下，左侧门框存在较为明显的变形，而动态全景图方案则较为正常。

改进 2：应用更精细的三角面片

猜测左侧门框处的形变是由三角面片过大，由全景球面不贴合导致失真而引起。考虑对三角面片作进一步细分，进行更加细致的纹理贴图来消减失真现象。对每一个三角形的每一条边，设 n 为边上节点数目，当 $n = 2$ 时，无需额外内插其他节点（如图 4.7(a)）；当 $n > 2$ 时，需要在三角面片边以及内部内插一些节点，并形成三角面片（在重心坐标系可方便地进行节点内插）。从图 4.7 可以看到，随着三角面片不断细分，面片更加贴合全景球，显得更精细。但经测试发现，从全景球球心处查看过渡效果时，仅在 $\alpha < 0.2$ ，视觉效果有细微改善，但左侧门框处依旧存在变形。

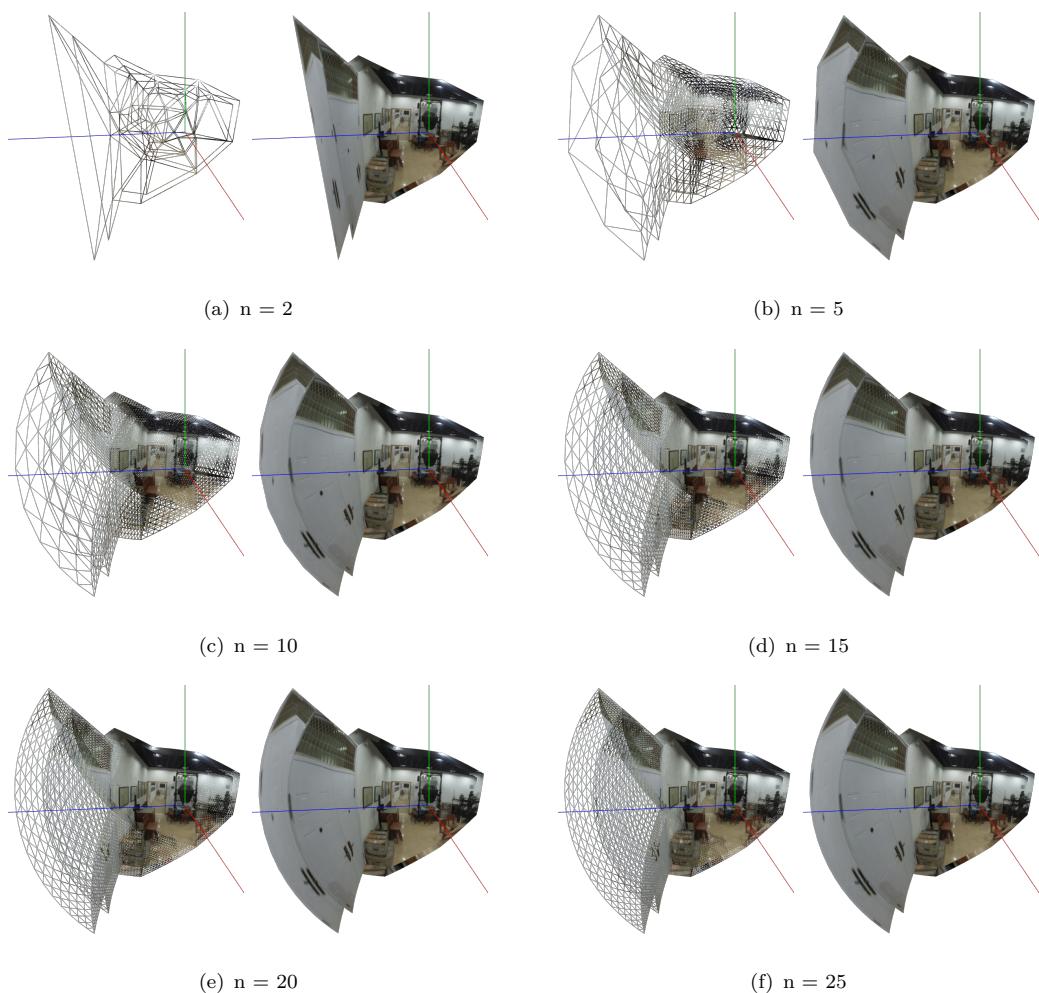
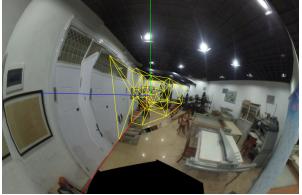
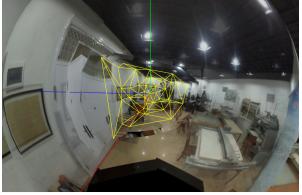
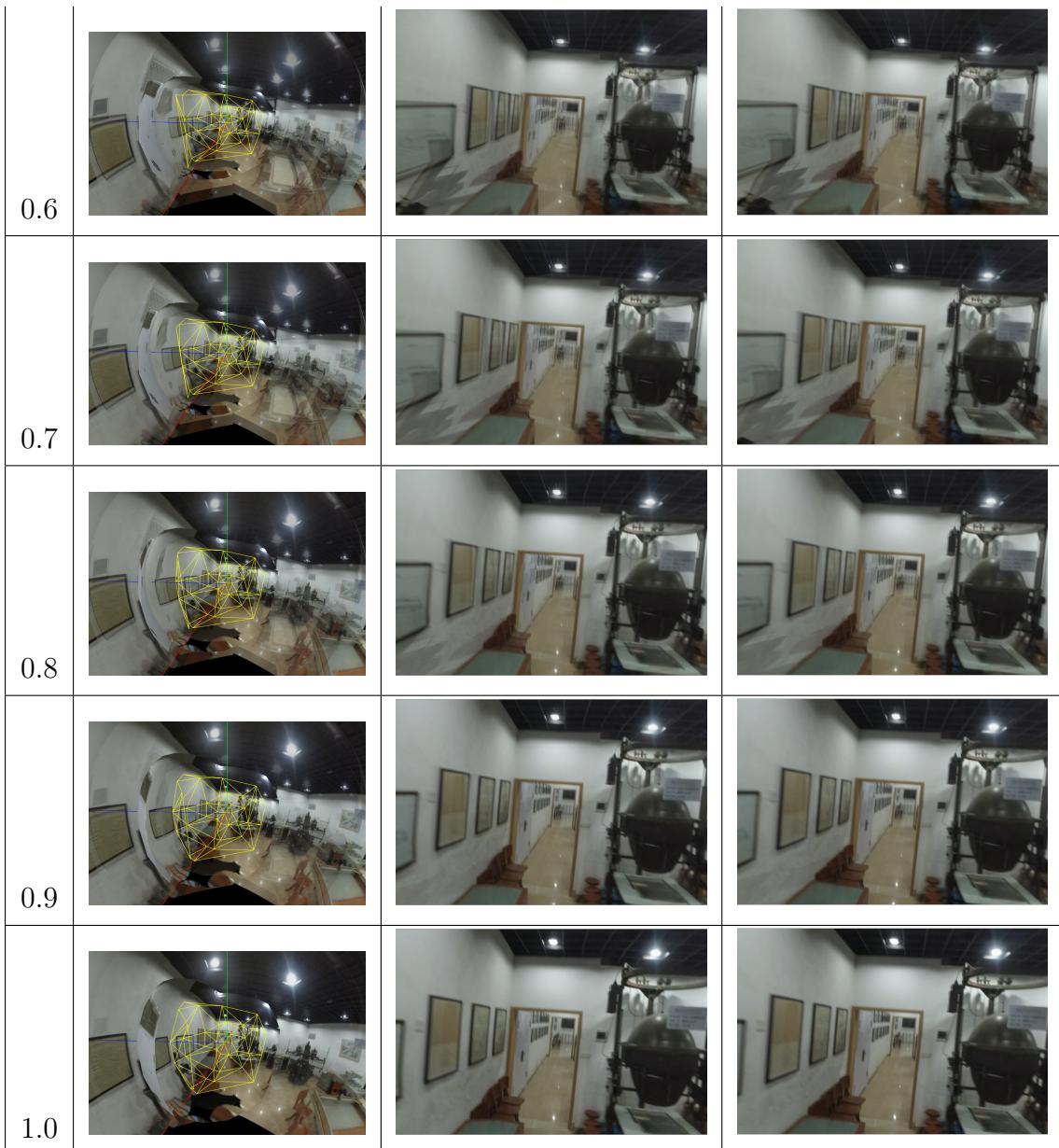


图 4.7 三角面片的细分

表 4.2 动态全景球效果

α	动态全景球 (示意)	动态全景球 (效果)	动态全景图 (效果)
0.0			
0.1			
0.2			
0.3			
0.4			
0.5			



改进 3：选定更合理的同名控制点

发现左侧门框变形位置是几个较“瘦”的三角形的汇集处。在这里，推测是因同名控制点对选取中存在的问题，导致 Delaunay 三角剖分算法未能避免“瘦”三角形的产生，局部才产生了畸变。如图 4.8，通过添加额外的控制点，使得三角化过程中，门框所在平面不再被三角形穿越（如图 4.8(b)），变形问题就消失了。可以看到，相比动态全景图方案，动态全景球方案对同名控制点对的选取更敏感，因同名点质量问题导致的不合理三角化对动态全景图方案影响较小，对动态全景球方案影响较大。动态全景球方案在解决动态全景图方案需要不断渲染动态全景纹理导致性能瓶颈问题，带来性能提升的同时，过渡效果也变差了一点。

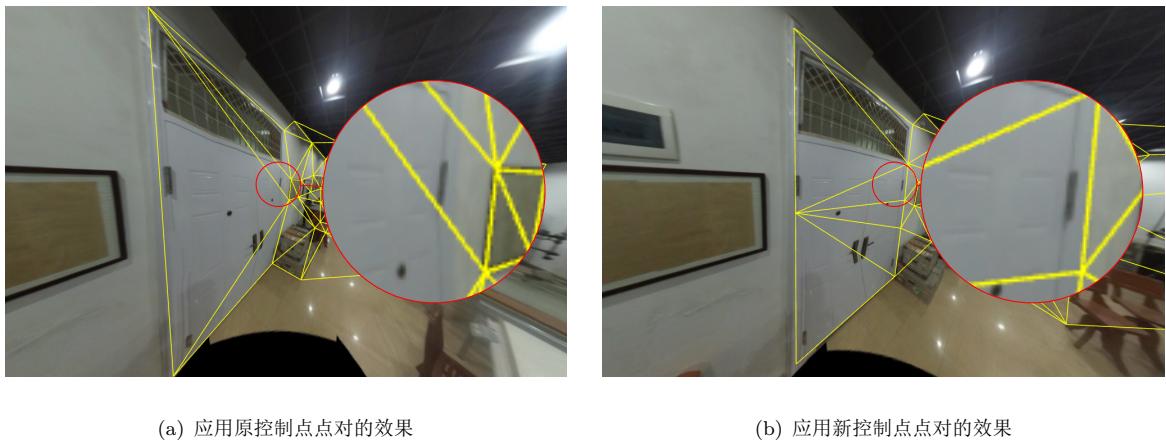


图 4.8 控制点点对影响动态全景球场景过渡方案的效果

4.3.2 效果与评价

动态全景图和动态全景球都实现了较为自然的全景场景过渡效果。两者原理基本一致，都需通过同名控制点点对指导三角面片的过渡和变形，前者通过“播放”动态全景图达到自然过渡，后者通过动态调整全景球以达到近似的效果。相比全景漫游系统中常见的过渡处理，这两种方案在视觉效果上具有明显优势。

相比动态全景球方案，动态全景图方案理论直观，易于实现。虽然动态全景纹理实时渲染会占用较大内存，影响过渡时期视图的重绘帧率，可能存在不自然乃至卡顿的现象，但若将纹理生成放到服务器后台完成，可减轻浏览器前端的计算和内存压力（但随之带来了网络带宽压力），还可以对用户隐藏全景自然场景过渡的实现方法，避免被抄袭。

相比动态全景图方案，动态全景球无需动态生成每一帧纹理贴图并重绘全景球，具有极大的性能优势，而效果与动态纹理方案基本一致。因无需渲染纹理，减小了一个渲染循环，动态全景球方案具有重绘帧率高，内存占用小的特点，场景过渡更流畅。但较动态全景图方案，动态全景球方案需要实时操作多个三角面片的空间位置、形状和纹理，三角面片顶点位置的插值还需要使用 SLERP，而不是简单的 LERP，导致动态全景球方案的代码实现难度更大。最后，动态全景球方案对同名控制点点对的选取更敏感。不恰当的控制点选取，可能导致场景中物体发生畸变，如扭曲的门框等，影响场景漫游站点切换时的视觉效果。

4.4 效率测试

从体验上，动态全景球方案确实比动态全景图更流畅。为定量分析两者的性能差异，现通过系统渲染压力测试来比较两者的效率。测试时使用同样的全景图和同样的同名控制点对，分别使用动态全景图方案和动态全景球方案不断更新 α 值，进行视图的重新渲染。如图 4.9，在微信内置的浏览器中，两种方案在当前站点环视漫游时，都可以维持 60 FPS (Frames Per Second) 的帧率。当点击“过渡”时，场景自然过渡到下一站点，使用动态全景图方案时，页面稍有卡顿；使用动态全景球时，页面十分流畅。点击控件栏上的“压力测试”，如图 4.9(a)，动态全景图方案下，重绘帧率急速下降到约 10 FPS，页面反应也变得迟钝起来；而采用动态全景球方案时，如图 4.9(b)，帧率只是稍微下降到约 50 FPS。如表 4.3，在 PC 端浏览器以及多种移动端浏览器中的测试结果也基本相同，都证实了动态全景图球方案相比动态全景图方案的性能提升。

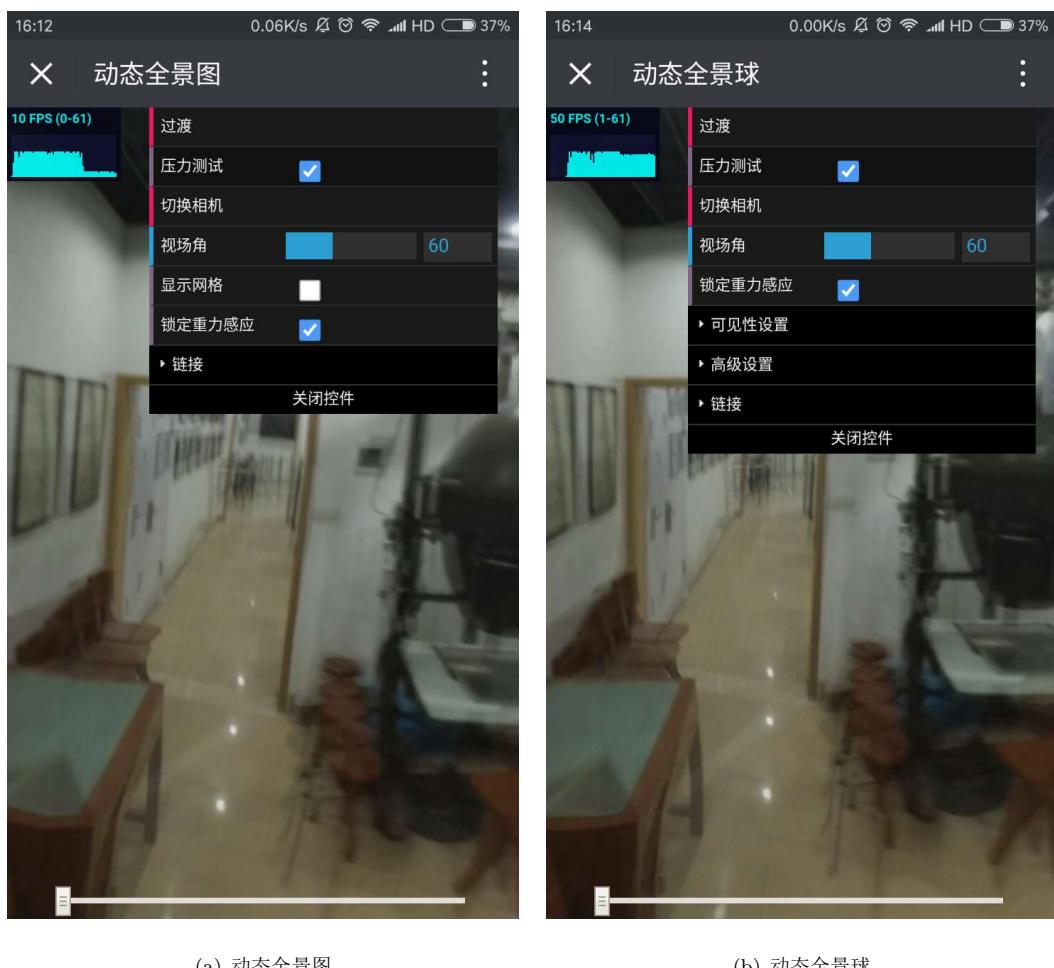


图 4.9 压力测试

表 4.3 帧率测试 (漫游/过渡/压测)

平台	动态全景图方案	动态全景球方案
Windows 10 – Firefox	58/01/01	56/44/45
Windows 10 – Chrome	60/18/15	60/56/54
Windows 10 – IE	无数据	无数据
Windows 10 – Edge	无数据	60/56/54
红米 Note 3 – 微信	60/18/18	56/55/56
红米 Note 3 – QQ	60/13/12	60/41/56

4.5 本章小结

本章先介绍了几种常见的室内全景漫游系统场景过渡方案：跳变、渐变、拉伸、干扰和视差法，并指出了它们各自的问题：i) 跳变法对用户体验的不重视；ii) 渐变法效果差让人产生空间迷失感；iii) 拉伸法视觉效果过于夸张不适合室内场景；iv) 干扰法治标不治本不是一个真正解决方案；v) 视差法效果虽好但需要真三维模型的支持。又在视差法的启发下提出了动态全景图和动态全景球这两种全景过渡方法，通过提取同名控制点生成三角网格引导图像变形，使得全景场景无缝过渡。实验结果表明，全景场景中的物体，无论是天花板上的灯、左侧的桌子、墙上的画幅还是右侧的测绘仪器，都能自然地向侧边缓缓移动，无缝过渡到新的位置上，同时视图自然地拉伸放大，给人以向前步行漫游的感觉。两种方案视觉效果基本一致，前者重绘效率低但理论直观、易于实现；后者对控制点敏感，但无需渲染全景图，重绘效率高，性能更好。

5 室内 360 度全景场景无缝漫游原型系统设计与实现

构建原型系统并作对比分析才能验证本论文提供的全景漫游过渡解决方案的有效性。本章主要介绍室内全景漫游原型系统搭建所依赖的软件系统，展示实验数据下取得的效果，并分享一点工程经验。

5.1 软件环境

本着 i) 简单专注；ii) 易扩展，便于后期整合 的实现原则，本室内全景漫游系统原型使用了一些成熟的跨平台解决方案：用 JavaScript 来做代码实现，用 Makefile 来做工程管理，用纯文本 (JSON 文件) 和全景图直接放文件系统来做数据存取，而不使用后端服务器和数据库。

5.1.1 基于 Three.js 的单页应用

JavaScript 诞生于 1996 年，是一门动态编程语言。JavaScript 被广泛使用在 HTML 网页中，用来给页面增加动态功能。与 C/C++ 这类静态、强类型语言不同的是，直译式脚本语言 JavaScript，是一种动态类型、弱类型、基于原型、表现力极强的编程语言。JavaScript 语法灵活，既有面向对象的特点，又算得上一门函数式编程语言。除此之外，JavaScript 还是浏览器普遍支持的编程语言，可用来实现跨平台的程序，免安装且能快速部署和更新。

WebGL 是基于 OpenGL ES 的 Javascript API，通过 GLSL 着色语言在 HTML5 里的 Canvas 页面元素上进行 GPU 加速了的三维绘图，可以看成 OpenGL 的浏览器版本。直接使用 WebGL 需要对 GLSL 着色语言有足够的了解，使用起来较为复杂，而且开发效率低。所以本原型系统的搭建使用了一个流行的基于 JavaScript 和 WebGL 的三维显示库：Three.js。Three.js 由 Ricardo Cabello 开发，利用 WebGL 进行浏览器内 3D 场景和模型的渲染。使用 Three.js 的简单步骤是：1. 新建一个场景 (Scene)；2. 向场景中添加一些三维物体 (Mesh)，比如室内全景系统中最主要的三维物体全景球就是一个贴了全景图的三维球体；3. 向场景中添加相机 (Camera)；4. 渲染器 (Renderer) 以相机做视口来渲染三维场景为二维图片，并呈现在页面中的 Canvas 元素上。使用 Three.js

可快速写出三维场景和模型渲染代码，而且还能有效利用 WebGL 加速，甚至在不支持 WebGL 的移动设备或是 WebView 控件下，还可以使用 CSS3D 渲染 (CSSRenderer) 或者 Canvas 的二维绘图方式 (CanvasRenderer) 代替 WebGL 渲染 (WebGLRenderer)。从开发难度、效率以及系统兼容性上考虑，使用 Three.js 都是实现本原型系统的不二之选。

值得一提的是，全景漫游领域内更常使用的是 Krpano 全景播放库。因其体积小巧、使用灵活和性能好，提供全景图的发布和交互功能以及强大的附加功能，如指南针、导航地图，又提供 Flash 版本和 JavaScript 版本两种选择，几乎是所有全景漫游系统工程实践中的首选。但 Krpano 使用 XML 进行配置和扩展，和普通的 JavaScript 开发差异较大；且相比 Three.js，Krpano 只是一个三维全景播放器，不是一个完备的三维库，不能方便地进行矩阵运算、三维物体的渲染，也不能方便地使用 GLSL 语言编写 Vertex Shader 和 Pixel Shader 来进行全景的特殊处理，而这都是实现一个完备的场景过渡自然的全景漫游系统所必要的。最后，Three.js 是托管在 GitHub 上的一个活跃的开源项目，遇到问题时能方便地从社区中寻求帮助，且它使用 MIT 协议，即使自己修改后的代码不开源或是作商用，也没有任何问题。综合上述考虑，本原型系统使用了 Three.js 而不是 Krpano。

其他的，jQuery 是流行的 JavaScript 库，主要用来操作 DOM 元素，进行事件绑定；dat.gui 可以方便地修改程序中的一些参数，是一个简单易用的输入控件库；tween.js 提供了变量在两个状态间进行渐变过渡的功能，主要用于控制场景过渡时的渐变程度值 α ；toastr.js 提供消息提示功能；OrbitControls.js 和 DeviceOrientationControls.js 分别提供了 PC 端的鼠标操作功能和移动端的重力感应操作功能这两种各自平台下直观的全景交互方式。

使用了如上的 Three.js、jQuery、dat.gui.js 等 JavaScript 库，原型系统最终以单页应用呈现。单页应用的好处是：i) 前后端分离，原型系统可暂时不实现后端服务器，仅使用文件系统来做数据存取，后期可在几乎不修改页面代码的情况下将本应用整合到后端服务器，并利用数据库存取数据；ii) 具有桌面应用的即时性、网站的可移植性和可访问性，同一套程序代码不用修改就可以用于 PC 浏览器、移动端浏览器以及微信平台，且无需用户安装；iii) 本地调试容易，开一个简单的文件服务器即可查看效果，无需编译运行，极大方便了原型系统的实现开发。可以说，直接在 HTML 页面中撰

写的 JavaScript 代码逻辑，几乎就是原型系统的全部。

5.1.2 用 Makefile 管理工程

Makefile 是类 Unix 环境下（比如 Linux）的一种自动化处理“脚本”文件，其基本语法是：“**目标** ←^{命令} **依赖**”。只有在目标文件不存在，或目标比依赖的文件更旧，命令才会被执行。使用跨平台的 make 来执行 Makefile 的逻辑，即可实现工程代码的管理。简单使用步骤为：i) 确定工程最终的输出文件（目标），比如本原型系统的主界面 index.html；ii) 确定目标的依赖项，如用到的 JavaScript 库和 main.js、main.css 等；iii) 明确从依赖项到目标的处理流程，如直接拷贝 index.html 到目标文件夹，压缩、混淆 JavaScript 代码后输出到目标文件。

通过 Makefile 写好了工程处理流程后，还可以监控文件夹下的相关文件的变动。当文件修改后自动重新 make，实时更新效果。

5.2 实验数据

如第 3 章所言，全景漫游系统的输入主要是全景图和三维模型数据。使用 Three.js 可以很方便地整合全景球和三维模型数据，将全景球放置到三维模型的参考框架下。但由于不可抗因素，暂时无法得到三维模型数据，只得从全景图中提取简单的包围盒模型以作三维参考框架。另一方面，实验用的全景图数据的质量不高，存在明显缺陷：i) 因拍摄时相机发生抖动，全景图存在鬼影（Ghost Effect），导致图像分辨率高但质量差^[7]；ii) 天顶方向不正，使得提取的包围盒存在明显变形；iii) 拍摄的环境不具有代表性，过渡的场景不够典型；iv) 采集的所有全景站点在一条直线上，没有多个方向的跳转。这些都对实验和测试造成了不少困扰。

从全景图 4.2(a) 中提取包围盒模型，需要找到全景图中八个墙角的坐标。一开始尝试通过线提取来寻找灭点的方法来确定墙角坐标（如图 5.1）^[2, 15, 22, 31, 33, 37]，但由于工作量巨大无法在短期内实现墙角点的自动提取，最后只得通过人工确定墙角坐标的方式半自动地生成包围盒模型。如图 5.2，生成包围盒模型的基本步骤为：i) 人工确定了八个墙角点坐标 p_i , $i \in \{1..8\}$ ；ii) 根据全景图天顶方向和相机在室内的高度相对比确定“地面”和“天花板”；iii) 将八个墙角点 p_i 投影到三维空间中的“地面”和“天花板”，对应 P_i , $i \in \{1..8\}$ ；iv) 根据 P_i 生成前、后、左、右、上、下六个平面，并

把全景图投影到这六个平面上，形成包围盒模型。可以看到，由于全景图是从一个中心点拍摄而成，形成的包围盒模型在三维环境中任意位置查看时可能存在特征明显的畸变，比起一般三维模型有较大程度的失真。但若从靠近全景球中心的位置查看，效果反而比三维模型好。

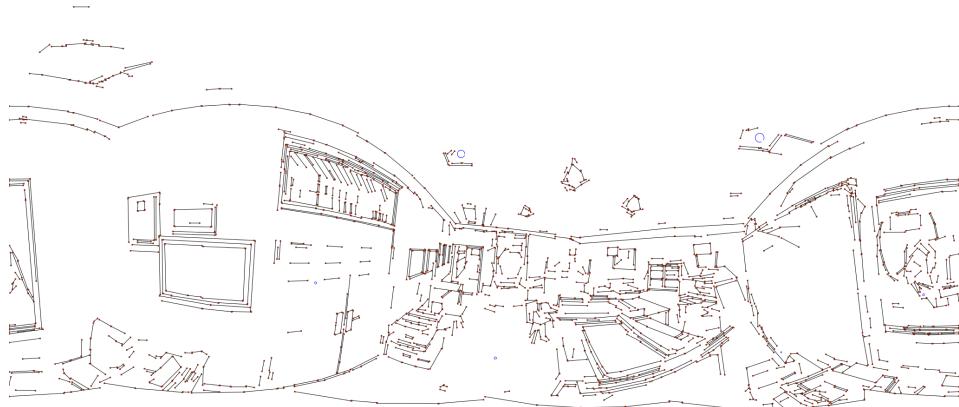


图 5.1 通过线提取来尝试找到墙角点

至于站点信息的组织，原型系统使用了最简单的 JSON 文本。其中记录了一个场景下各个全景站点的信息，包括但不限于：i) ID；ii) 名称；iii) 简介；iv) 全景图路径；v) 全景姿态信息（主要是北方向和天顶方向）；vi) 可跳转到的其他漫游点。

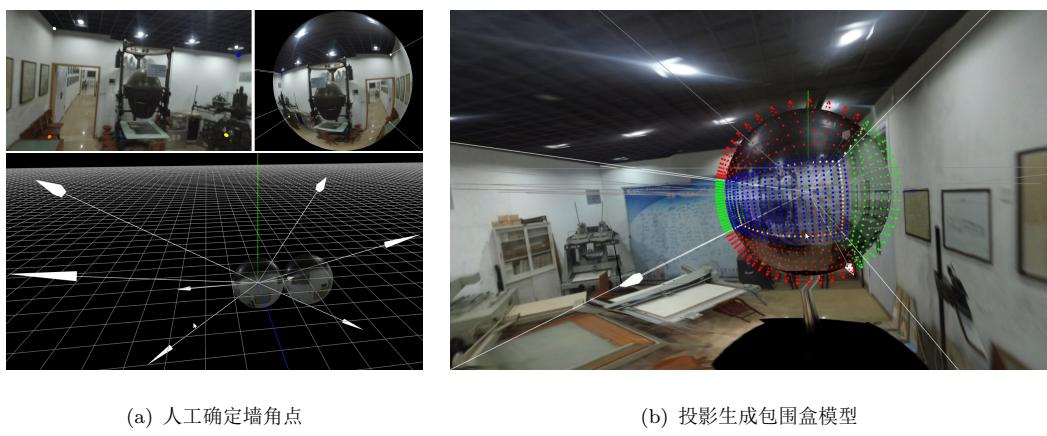


图 5.2 从全景图生成伪模型数据

综上所述，360 度全景无缝场景过渡的室内全景漫游系统的数据处理流程图见图 5.3。主要包括 i) 使用 PTGui 将原始影像拼接成使用 Plate Carrée 投影的宽高比为 2:1 的全景图；ii) 人工定位全景图中墙角点，生成包围盒模型（因为缺乏三维数据），作漫

游环境模型; iii) 人工确定可漫游站点间的同名控制点对, 写入配置文件。这样, 一系列原始影像变成了可供 360 度全景漫游系统前端使用的全景漫游数据。在漫游站点载入全景图, 便可以自如浏览全景站点; 跳出全景, 还可以浏览漫游点的三维环境; 在全景站点间, 还可载入切换配置 (主要是同名控制点对) 并通过动态全景纹理方案或者动态全景球方案实现全景场景的无缝自然过渡。

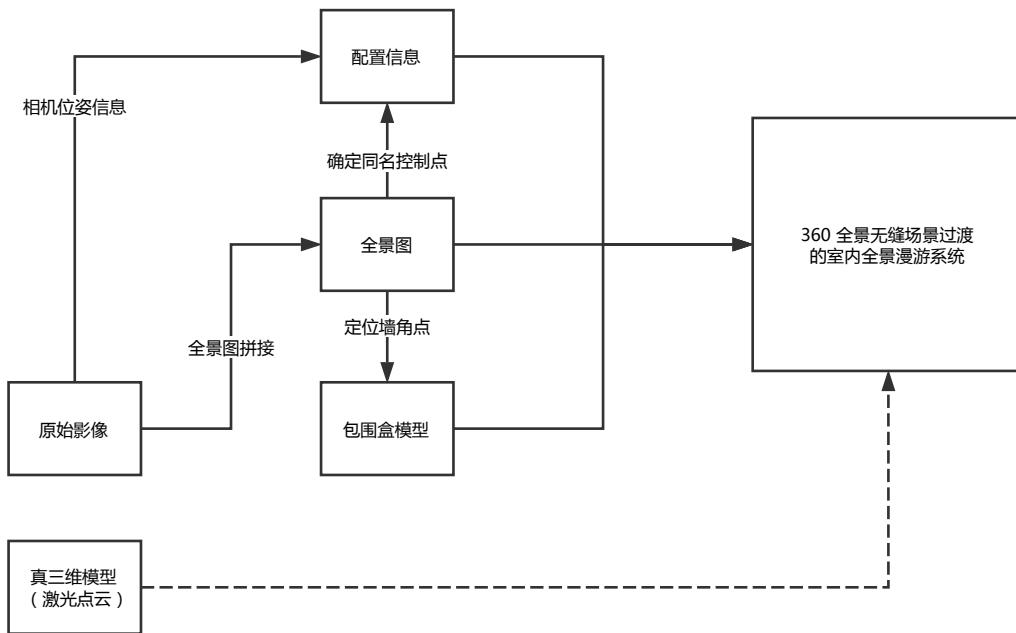


图 5.3 处理流程图

5.3 实现的功能点

室内 360 度全景无缝漫游原型系统已实现如下功能点:

1) 全景图的加载

如图 5.4(a), 可以通过页面控件上的“选择全景图”加载本地全景图片, 还可以通过“切换全景图”载入预置的几张全景漫游站点。载入的图片默认通过 Plate Carrée 投影在全景球上。在 PC 端, 还可以通过拖放的方式来加载本地全景图片。考虑到软件的鲁棒性, 即使图片不满足宽高比为 2:1, 全景加载也会完成。

2) 全景场景的浏览操作

加载后的全景图呈现拍摄处的全景场景供用户漫游。如图 5.4(b), 可以通过页面上的控件直接修改观察相机的观察方向 (相机视线中心的 lon、lat 值), 还可以修改观

察相机的视场角。除了通过控件来调整漫游观察相机，还可以通过鼠标或者重力感应传感器。在 PC 端，通过鼠标拖动的方式改变观察方向，通过鼠标滚轮调整视场角大小；在移动端，通过重力传感器感知观察方向，通过触屏手势调整视场角。两种交互方式简单直观，易于上手，为全景浏览器提供了极大便利。除此，如图 5.4(c)，在 PC 端上还可以获取全景图中鼠标当前位置的经纬坐标，这样一来，场景中添加热点后可直接利用这一接口进行交互，方便系统的扩展。



图 5.4 全景漫游系统实现的功能

3) 场景的浏览交互

当用户跳出全景站点的环视漫游时，漫游视图逐步切换到漫游点的整体场景，用户可以浏览真实三维环境，以整体把握各全景站点的空间分布。用户可以通过鼠标和触屏操作来观察场景模型，还可以通过点击或轻触全景球，自然过渡到某一具体全景站点的环视漫游。

4) 全景漫游站点之间自然过渡

在某一全景站点的环视漫游中，如果当前方向存在可以漫游的临近漫游点，视图中会显示一个全景切换按钮。用户点击按钮后，场景通过动态全景球方案在 1-3 秒内自然地过渡到下一漫游点，感觉起来就像用户在真实三维环境中前进走到下一个漫游位置一样。在页面下方还有图片轮播列表，通过点击图片可进入相应全景漫游站点。如果当前漫游点与欲漫游全景站点相邻，观察相机会自动旋转到相应视角并通过动态全景图或动态全景球方案自然过渡视图，如果当前漫游站点和欲漫游全景站点不相邻，视图转入临时观察相机，进入模型视角，旋转到适当的位置后进入新的全景漫游点。

5.4 工程上的一些考虑

5.4.1 全景图的压缩

优化全景图数据的体积是全景系统中极有意义的一环。在不降低图片质量的情况下减小图片文件的大小，就能提供更流畅的加载。

全景图通常以 JPEG 格式存储以降低文件大小。JPEG 使用的是一种基于离散余弦变换的有损压缩方法，不同的“配置”会对图像大小产生较大影响，因此在保存全景图时，可通过设定 JPEG 保存的参数来压缩全景图容量。首先，JPEG 图片有两种保存方式，分别被称为 Baseline JPEG (标准型) 和 Progressive JPEG (渐进式)。浏览器中，两者显示方式不同。如图 5.5(a) 和图 5.5(b)，前者一行一行自上而下显示，后者先显示模糊的图片，再逐渐清晰。测试发现同一图像分别存储成 Progressive JPEG 和 Baseline JPEG 文件格式，两者文件大小的差值与图像尺寸基本呈线性相关，在作简单的线性回归后发现：i) 当原图片大概在 10 KB 左右时，两种保存方式下输出图片的文件大小近似；ii) 当原图片大于 10 KB 时，采用 Progressive 方式保存时文件更小（大概是 94% 的可能性）^[26]。因全景图片的尺寸都显著的大于 10 KB，因此可以 Progressive 保存方式来降低文件大小。其次，模糊的图片有更高的压缩率，适当对原图进行高斯模糊，在同等设置的情况下能显著降低文件大小，这尤其适合图片本身质量不高的情况。最后，JPEG 图片中可能存在注释或 Exif 标签信息（记录数码照片的属性信息、拍摄数据等），这些信息无益于纹理贴图，也可直接去除。

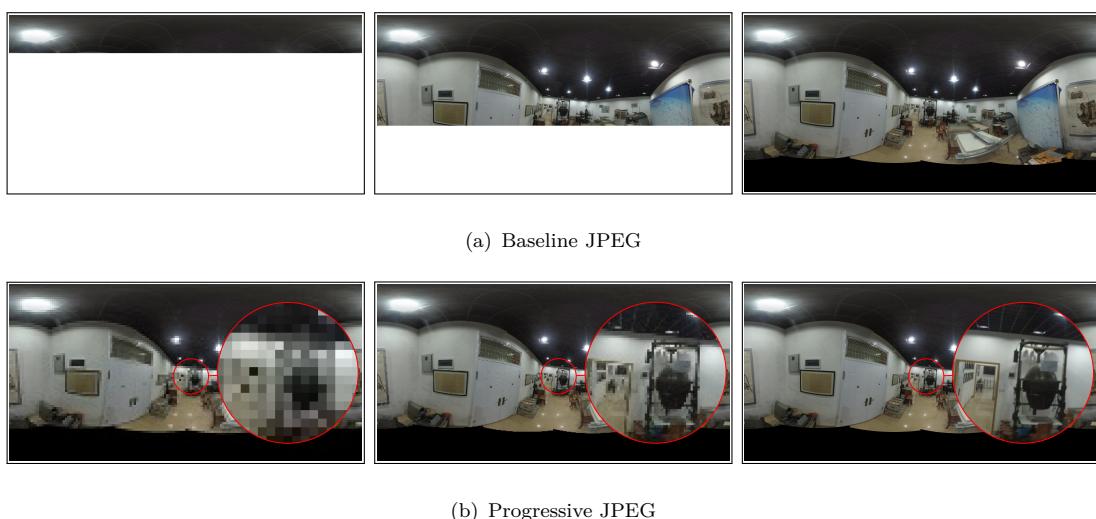


图 5.5 浏览器中 Baseline JPEG 和 Progressive JPEG 图片加载效果

综合上述考虑, 使用图像处理工具 ImageMagick 对全景图批处理以降低文件大小。经代码 5.1 压缩后的全景图由原来的 13.3 MB (分辨率为 8192×4096), 降低为 1.88 MB, 图片质量的损失肉眼不可分辨。进一步将图片降采样到 2048×1024 , 文件大小降至 184 KB, 全景查看时依旧没有肉眼可见的质量损失。

代码 5.1 压缩低质量 JPEG 图像

```

1 #!/bin/bash
2
3 convert \
4   -strip \
5   -interlace Plane \
6   -gaussian-blur 0.05 \
7   -filter Lanczos \
8   -quality 85% \
9   $1 ${1%.}._minified.jpg

```

5.4.2 全景图的分层切片后确保代码的可复用性

在一些对全景场景分辨率要求较高的应用场景下, 不宜通过降低全景图质量、降低文件大小的方法来提高全景纹理的加载速度和全景渲染的流畅性。此时应当采集、拼接出更高分辨率的原始全景图, 并进行分层、切片处理。在浏览全景站点时, 先加载较为模糊的全景图片, 再逐步加载适合当前视域的更清晰的图片切片。这样的策略势必影响全景查看工具的代码实现。一旦原始全景图分辨率调整了, 或者分层、切片的方式变化了, 就需要相应地修改全景查看工具的源码: 重新计算每个切片对应的球面的范围, 重新贴图并渲染。如此高度的耦合性显然不利于代码的集成和项目的维护。

换一个思路, 不再考虑如何把新的切片更新到全景球的新的位置上, 而是考虑如何生成一个新的全景图。通过 Three.js 的 Canvas Renderer, 可以将 HTML 页面上的 Canvas 元素作为纹理贴图到全景球上。只要在获得新的切片后更新 Canvas 元素上的“虚拟全景图”, 再整体进行渲染重绘, 便能达到更新全景球上一部分球面的效果。具体操作步骤为: i) 撰写脚本根据一定的规则对一张高清全景图分层、切片, 得到一系列图; ii) 根据上面的规则撰写对应的“虚拟全景图”适配代码, 负责一个 Canvas 元素的绘制和更新; iii) 将用户的视域变化绑定到适配器上, 根据视域的变化适时地更新“虚拟全景图”。

5.5 本章小结

本章简要介绍了在简单专注、易于扩展的实现原则下，通过 WebGL 来实现 360 全景无缝场景过渡的室内漫游原型系统。先后介绍了浏览器中的编程语言 JavaScript，浏览器中的三维绘图接口 WebGL，流行的 JavaScript 三维库 Three.js，并在此基础上介绍了原型系统的搭建：通过 Makefile 来管理单页应用。接着介绍了实验数据，并提出一种在三维模型数据缺失的情况下从全景图中生成包围盒模型的方法。在给出总体操作流程并作小结后，又介绍了工程实践上全景图的压缩处理方法，全景图切片的加载和代码解耦的策略。

6 总结与展望

6.1 内容总结

全景漫游系统是一种流行的初级虚拟现实技术，因其成本低、视效好而应用广泛。流行的同时，一般的室内全景漫游系统都未能很好地解决漫游切换跳转时不自然的现象，场景切换问题成为了全景漫游系统的短板与不足。针对室内场景封闭性和局部性特点，本文提出了 360 度无缝场景过渡方案，分别用于在三维模型和全景之间的过渡、全景站点之间的过渡。实验表明，本文提出的内插临时观察相机位姿、动态全景图和动态全景球方案可以有效解决上述问题。

第一章绪论，先介绍了全景漫游系统及其研究现状，并指出其中普遍存在的过渡不自然的问题，继而引出本文主要内容及组织结构。第二章是对室内全景漫游系统相关理论知识、关键技术的介绍。主要包括三块：i) 全景图及其投影；ii) 数学基础；以及 iii) 模型数据与全景数据的采集和处理。因全景图投影对制定合理有效的影像数据采集计划有极大指导意义，所以首先介绍了全景图投影。第一部分在简要介绍全景图相关的几种常见的投影方式的同时，通过对比发现 Plate Carrée 投影的优势。第二部分有关数学的内容主要介绍了漫游系统中常用的多种坐标系统、三维物体姿态的多种表示法，以及模型数据和全景数据的采集和处理流程。

接下来，第三章和第四章分别解决三维模型与全景图的无缝过渡问题和全景站点之间的无缝过渡问题。前者通过引入额外的临时观察相机，进行初始状态到目标状态的线性、非线性内插来达到衔接两种异质交互方式的目的。后者通过引入同名控制点引导形变过程使视觉效果更自然。动态全景图方案效果逼真自然，但存在性能瓶颈，导致过渡时视图渲染更新帧率不够，可能存在卡顿问题；动态全景球方案效果与动态全景图方案近似，对同名控制点的选取敏感，但性能好，使用这一方案来过渡全景视图，渲染帧率几乎没有下降，过渡效果更流畅。

第五章介绍原型系统的实现，先提出原型系统的实现原则，并在此原则指导下又介绍了基于 Three.js 三维库的浏览器内解决方案。第六章总结与展望，总结在室内全景漫游场景过渡问题上做的一些思考、完成的工作，并介绍下一步可探索的方向以及一些可能的思路。

6.2 论文创新点

- 1) 混合组织全景数据和三维模型数据，将三维场景模型引入漫游系统；
- 2) 将颜色混合策略应用于前后视图两两匹配的三角面片而不是没有配准的整个视图，解决了场景过渡不自然的问题；
- 3) 实现了一个可用的原型系统，配置性高，按照预定的规则生成配置信息，即可方便地调用前端渲染模块来体验漫游环境。

6.3 工作展望

在三维与二维融合问题上，仅提出采用临时观察相机来作衔接这一策略，但何种内插方式可以提供更好的视觉效果，这个问题并没有很好地展开。在全景站点之间的自然过渡问题上，提出了动态全景图和动态全景球两种有效的过渡方案，但它们都依赖于人工提取的同名控制点，不同数量的控制点以及不同原则下选取的控制点对场景切换有何种影响，这一问题也有待考量。同时，在明确了应以何种原则采集何种数目的同名控制点后，如何通过特征点提取的方式自动获取优质的同名控制点又是另一个有趣的课题。

在基于全景图的室内全景漫游系统上研究得越多，越发觉解决无缝过渡这一课题的重要意义。尤其在小规模的漫游系统中，比如商铺浏览这一领域，自然的场景过渡具有极大价值。在这样的应用场景下，自动批处理提取有效的同名控制点不再必要；相比较而言，获取更优质的同名控制点（哪怕通过人工方式）以获得更流畅的漫游体验才是更关键的内容。因此本文提出的动态全景图和动态全景球方案具有现实的经济价值。

参考文献

- [1] M. Aly and J. Y. Bouguet. Street view goes indoors: Automatic pose estimation from uncalibrated unordered spherical panoramas. *Proceedings of IEEE Workshop on Applications of Computer Vision*, pages 1–8, 2012.
- [2] A. Ansar and K. Daniilidis. Linear pose estimation from points or lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):578–589, 2003.
- [3] F. Aurenhammer. Voronoi Diagrams —A Survey of a Fundamental Data Structure. *ACM Computing Surveys*, 23(3):345–405, 1991.
- [4] M. Bleyer and M. Gelautz. Graph-cut-based stereo matching using image segmentation with symmetrical treatment of occlusions. *Signal Processing: Image Communication*, 22(2):127–143, 2007.
- [5] S. E. Chen. QuickTime VR—An Image-Based Approach to Virtual Environment Navigation. *Computer Graphics*, 29:29–38, 1995.
- [6] J. Choi. Bézier Curve for Trajectory Guidance. *World Congress on Engineering and Computer Science*, pages 0–5, 2008.
- [7] J. Chon, E. Shimizu, and R. Shibasaki. 3D panoramic mosaicking to suppress the ghost effect at far-range scene for urban area visualization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3980 LNCS:261–267, 2006.
- [8] T. S. Collett. Insect Navigation: Visual Panoramas and the Sky Compass. *Current Biology*, 18(22):R1058–R1061, nov 2008.
- [9] Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi Tessellations: Applications and Algorithms. *SIAM Review*, 41(4):637–676, 1999.
- [10] Y. Furukawa, B. Curless, S. Seitz, and R. Szeliski. Manhattan-world stereo. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1422–1429. IEEE, jun 2009.
- [11] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Reconstructing building interiors from images. In *2009 IEEE 12th International Conference on Computer Vision*, number Iccv, pages 80–87. IEEE, sep 2009.
- [12] C. Geyer and K. Daniilidis. Catadioptric Projective Geometry. *International Journal of Computer Vision*, 45(3):223–243, 2001.
- [13] D. Gledhill, G. Y. Tian, D. Taylor, and D. Clarke. Panoramic imaging - A review. *Computers and Graphics (Pergamon)*, 27(3):435–445, 2003.

- [14] R. Gopavaram. Real Time Transferring of Panorama Images from a Spherical Camera System to Smart Phones. 1(April):1–42, 2013.
- [15] R. Grompone von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. LSD: a Line Segment Detector. *Image Processing On Line*, 2:35–55, 2012.
- [16] W. Heidrich. Computing the Barycentric Coordinates of a Projected Point. *Journal of Graphics, GPU, and Game Tools*, 10(3):9–12, 2005.
- [17] Y. Horrry, K.-I. Anjyo, and K. Arai. Tour into the picture: using a spidery mesh interface to make animation from a single image. *Proceedings of SIGGRAPH*, pages 225–232, 1997.
- [18] B. Jenny, B. Šavrič, and J. Liem. Real-time raster projection for web maps. *International Journal of Digital Earth*, 9(3):215–229, 2016.
- [19] J. Kessenich, D. Baldwin, and R. Rost. The OpenGL® Shading Language. *Language*, 1:1–29, 2010.
- [20] I.-C. Lee and F. Tsai. Applications of Panoramic Images: From 720° Panorama To Interior 3D Models of Augmented Reality. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-4/W5(May):189–192, 2015.
- [21] L. McMillan. An image-based approach to three-dimensional computer graphics. *Thesis (Ph. D.)–University of North Carolina at Chapel Hill*, 1997.
- [22] F. Moreno-Noguer, V. Lepetit, and P. Fua. Accurate non-iterative $O(n)$ solution to the pnp problem. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, 2007.
- [23] Y. Morvan and C. O’Sullivan. Handling occluders in transitions from panoramic images. *ACM Transactions on Applied Perception*, 6(4):1–15, 2009.
- [24] S. Peleg and M. Ben-Ezra. Stereo panorama with a single camera. *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, 1:395–401, 1999.
- [25] K. Petrinec and Z. Kovacic. The application of spline functions and bezier curves to AGV path planning. *IEEE International Symposium on Industrial Electronics 2005, ISIE 2005, June 20, 2005 - June 23, 2005*, IV:1453–1458, 2005.
- [26] A. M. Raid, W. M. Khedr, M. A. El-dosuky, and W. Ahmed. Jpeg Image Compression Using Discrete Cosine Transform - A Survey. *International Journal of Computer Science & Engineering Survey*, 5(2):39–47, 2014.
- [27] D. Ramey, L. Rose, and L. Tyerman. Alias/Waverfront MTL File Format, 2011.
- [28] N. Snavely, R. Garg, S. M. Seitz, and R. Szeliski. Finding paths through the world’s photos. *ACM Transactions on Graphics*, 27(3):1, 2008.

- [29] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring Photo Collections in 3D. *ACM Transactions on Graphics*, 25(3):835–846, 2006.
- [30] T. Stich, C. Linz, C. Wallraven, D. Cunningham, and M. Magnor. Perception-motivated interpolation of image sequences. *ACM Transactions on Applied Perception*, 8(2):1–25, 2011.
- [31] J. Tavares. A new approach for merging edge line segments. *RecPad95*, 1995.
- [32] M. Trapp and J. Dollner. A generalization approach for 3d viewing deformations of single-center projections. *International Conference on Computer Graphics Theory and Applications*, pages 163–170, 2008.
- [33] G. Wang, J. Wu, and Z. Ji. Single view based pose estimation from circle or parallel lines. *Pattern Recognition Letters*, 29(7):977–985, 2008.
- [34] M. Wang and S. Member. Panorama Painting: With a Bare Digital Camera. In *International Conference on Image and Graphics*, pages 82–86, 2009.
- [35] J. Warren. Barycentric Coordinates for Convex Polytopes. *Advances in Computational Mathematics*, 1(July):1–14, 1996.
- [36] D.-l. Way and J.-w. Shieh. A New Image Morphing Technique for Smooth Vista Transitions in Panoramic Image-Based Virtual Environment. *Transition*, 1998.
- [37] J. Xiao and Y. Furukawa. Reconstructing the World’s Museums. *Eccv*, pages 668–681, 2012.
- [38] Y. Xiong and K. Pulli. Fast Panorama Stitching for High-Quality Panoramic Images on Mobile Phones. *IEEE Transactions on Consumer Electronics*, 56(2):298–306, 2010.
- [39] G. J. Yang and B. W. Choi. Smooth trajectory planning along Bezier curve for mobile robots with velocity constraints. *International Journal of Control and Automation*, 6(2):225–234, 2013.
- [40] F. Zhang. Quaternions and matrices of quaternions. *Linear Algebra and its Applications*, 251:21–57, 1997.
- [41] Z. Zhu and A. R. Hanson. 3D LAMP: a new layered panoramic representation. *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, 2:723–730 vol.2, 2001.
- [42] 党建武. 基于图像转换的全景平滑漫游技术. *计算机工程与设计*, 36(5), 2015.
- [43] 李怡静. 多视点全景图与平滑漫游的研究与实现. *计算机工程*, 35(12), 2009.
- [44] 韦群. 基于立方体全景图的虚拟场景浏览技术研究及实现. *中国图象图形学报*, 1(09):94–99, 2003.

致 谢

感谢我的导师胡庆武老师的耐心和宽容；感谢涂金戈、赵鹏程等同学的陪伴；感谢张媛细致的审校。毕业之际，分享一句话：

“In theory, there is no difference between theory and practice.

But, in practice, there is.”

希望毕业后的自己不要成为一个眼高手低的人。

本人的研究成果是在前人的基础上取得的，他们的许多研究思路和研究方法为本文的顺利开展提供了宝贵的借鉴资料。在此，向直接或是间接给予我启迪的各位前辈和同行表示衷心的谢意。