

专注于 3D地形 编程

特伦特·波拉克



© 2003 年,Premier Press,Course Technology 的一个部门。版权所有。未经 Premier Press 书面许可,不得以任何形式或任何方式 (包括影印、录音或任何信息存储或检索系统)以任何形式或任何方式 (包括电子或机械)复制或传播本书的任何部分,除非在回顾。



Premier Press 标志和相关商业外观是 Premier Press 的商标,未经书面许可不得使用。所有其他商标均为其各自所有者的财产。

出版商: Stacy L. Hiquet 营销

经理: Heather Hurley Acquisitions 编

辑: Emi Smith 项目编辑/文案编辑: Karen

A. Gill 技术审阅: André LaMothe

室内布局:肖恩晨星

封面设计: Mike Tanamachi

索引器:雪利酒梅西

校对:珍妮戴维森

Premier Press 标志和相关商业外观是 Premier Press 的商标,未经书面许可不得使用。所有其他商标均为其各自所有者的财产。

Black and White 是 Black and White Studios 的商标。Direct3D 和 Microsoft Visual C++ 是 Microsoft Corporation 的商标。Starsiege: Tribes 是 Sierra Entertainment, Inc. 的商标。TreadMarks 是 Longbow Digital Arts 的商标。

所有其他商标均为其各自所有者的财产。

重要提示: Premier Press 无法提供软件支持。请联系相应软件制造商的技术支持热线或网站寻求帮助。

Premier Press 和作者在本书中试图通过遵循制造商使用的大写风格来区分专有商标和描述性术语。

本书中包含的信息由 Premier Press 从被认为可靠的来源获得。但是,由于我们的消息来源、Premier Press 或其他人可能会出现人为或机械错误,出版商不保证任何信息的准确性、充分性或完整性,并且不对任何错误或遗漏或从中获得的结果负责使用此类信息。读者应该特别意识到互联网是一个不断变化的实体。

自从本书出版以来,一些事实可能已经改变。

国际标准书号:1-59200-028-2

美国国会图书馆目录卡号:2002111228

在美利坚合众国印刷

03 04 05 06 BH 10 9 8 7 6 5 4 3 2 1

Premier Press,课程技术的一个部门
2645 伊利大道,套房 41
俄亥俄州辛辛那提 45208

前言

你看过多少次你最喜欢的节目地形渲染算法的大量帖子似乎从各个角度向您飞来飞去?地形渲染似乎是当今业余程序员最喜欢的主题;它是一个很好的门户,可以解决更苛刻的问题及其解决方案。然而,地形渲染绝不是一个简单的问题,一个特定的解决方案可能会变得相当复杂。来自“编程”生活的各行各业的人们已经尝试提出下一个最佳解决方案来呈现他们对完美世界的想法。甚至有人敢说,有多少写地形引擎的人就有多少地形渲染算法。这些解决方案中的大多数是更广泛接受的解决方案的变体。这些解决方案通常被人们接受为具有良好性能的解决方案。其中一些已经存在了很长一段时间,并且多年来一直在进行修改,以适应它们要运行的不断变化的硬件。

本书采用了其中三个普遍接受的解决方案,并逐步完成了它们。我很高兴地说,其中一个解决方案来自我自己的编程思想水坑,我在顿悟中不时冒出。

由于本书比较了三种地形渲染解决方案,没有任何偏见或偏见,显然我不是一个谈论它们的人。我会留给作者吧。

威廉·H·德波尔

致谢

哇,需要感谢的人太多了。让我们从标志性人物开始:我的家人和朋友。首先,我要感谢我的妈妈、我的继父、我的妹妹和我的狗。他们摇滚,没有他们,我现在不会写这篇文章。我还要感谢我的朋友们。感谢 Kyle Way (你欠我的)在深夜与我打交道,并总是让我分心。感谢 Nate、Renae 和 Randy 一直都在。

还要感谢 Luke、Claudia、Amanda、Laurelin、Marissa、Ella、Rebecca、Lacee、Laura 和其他所有人。你知道你是谁!

接下来是感谢我不那么亲近的人的时候了。(事实上,这些人都是好朋友,但我以前从未真正见过他们。)首先,感谢 Evan 是一个很好的编程伙伴和好朋友。还要感谢 Ron 帮了我很多忙,感谢他成为一个好人。另外,非常感谢 Dave Astle、Kevin Hawkins 和 Jeff Molofee 让我走到了今天,并成为了伟大的导师。还要感谢 Mike、Sean 和 Warren 是三个非常酷的家伙。我还要非常感谢#gamedev 中的所有家伙,他们忍受了我深夜关于绿色兔子和猫在我的显示器周围跑来跑去为圣诞灯火发动战争的咆哮。

现在,我要感谢所有使这本书成为可能的人。非常感谢 Emi Smith、André LaMothe 和 Karen Gill 帮助使这本书成为最好的,实际上给了我写它的机会,而且通常是一群很棒的人。我还要再次非常感谢 Willem de Boer、Stefan Rottger 和 Mark Duchaineau 为我提供了想法并审查了各自的章节以使它们成为最好的。此外,感谢 Longbow Digital Arts 的人让我在本书的 CD 上使用TreadMarks的演示,感谢 Digital Awe 的人让我将热带风暴放在 CD 上,以及其他让我使用他们的本书 CD 上的演示:Thatcher Ulrich、Leonardo Boselli 和 Serba Andrey。

最后,我要感谢那些根本不知道自己与本书有关的人和事。非常感谢 Fender 制作了我令人难以置信的吉他。此外,百事可乐公司制作传奇的激浪也值得一两次感谢。另一个非常感谢 New Found Glory 的人,他们是一支伟大的乐队,也感谢 Tool、The Offspring、Blink 182 和 Nirvana。最后,感谢 Pacific Sun 的员工为我提供了“编码服装”。

关于作者

Trent Polack是一名高中生,目前就读于 Kalkaska 高中。他从 9 岁起就开始使用各种语言进行编程,当时他的表弟向他展示了 QBASIC 的乐趣。

Trent 是编程社区的积极贡献者,为 GameDev.net 及其附属机构贡献了一些教程。他对游戏编程充满热情,即图形、数据库,最重要的是引擎编码。当 Trent 不编程时,他对阅读(尤其是已故道格拉斯·亚当斯的书)、篮球、弹吉他和听音乐非常感兴趣。

内容一览

系列编辑的来信。十二至十三

介绍。十四

第一部分地 形编程介绍。 . 1

第 1 章户外之旅。 . 3

第2章地形101。 15

第 3 章纹理地形。 39

第 4 章照明地形。 57

第二部分高 级地形编程。 ... 73

第 5 章 Geomipmapping for

CLOD受损。 75

第 6 章攀登四叉树。 105

第 7 章 你可以在哪里漫游。 127

第 8 章 总结:

特殊效果等等。 165

附录 CD 上的内容。 205

指数。 209

内容

介绍。 十四

第一部分

地形编程简介。 . 1

第一章进入之旅

伟大的户外活动。 . 3

“地形?不,谢谢,我已经吃过了。”	4
一般应用。	4
地形和游戏开发。	5
演示构建变得简单!	8
主要演示。	8
本书一目了然。	11
第一部分:地形编程简介。	11
第二部分:高级地形编程。	12
演示。	12
概括。	13

第2章地形101。 15

高度图。	16
创建基础地形类。	18
加载和卸载高度图。	21
物质的蛮力。	24
分形地形生成。	27
断层形成。	27
中点位移。	33
概括。	37
参考。	37

第 3 章纹理地形。 39

简单的纹理映射。	40
程序纹理生成。	43
区域系统。	44
瓷砖系统。	46
创建纹理数据。	48
改进纹理生成器。	49
使用详细地图。	52
概括。	55
参考。	55

第 4 章照明地形。 57

基于高度的照明。	58
给光源着色。	59
五金照明。	62
光照贴图。	63
斜坡照明。	66
好的,斜坡照明很酷,但它是如何执行的?	66
创建斜坡照明系统。	68
动态创建光照贴图。	68
概括。	72
参考。	72

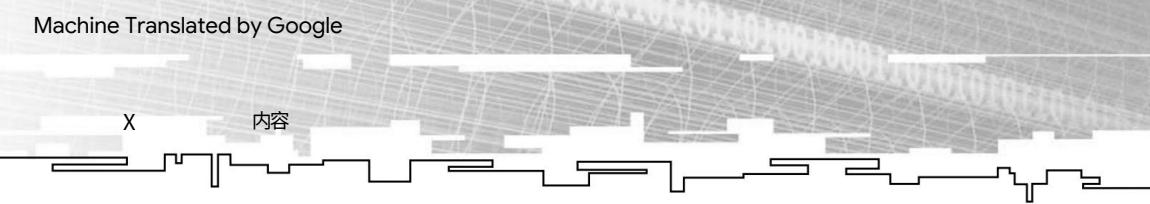
第二部分

高级地形编程。 73

第 5 章Geomip-mapping for

CLOD受损。 . . 75

CLOD 地形 101.	76
为什么要打扰 CLOD Terrain?	76
在 CLOD Terrain 的土地上,并非一切都是幸福的。	78
结束您对 CLOD Terrain 的介绍。	78
半 CLOD 受损者的 Geomipmapping 理论。	79
简单的基础知识。	79
三角排列变得容易。	80



X 内容

为非常轻微的 CLOD 受损实施 Geomipmapping。	87
修补它。	87
创建基本 Geomipmapping 实现。 ...	87
存在的问题有待解决。	98
概括。	104
参考。	104

第 6 章爬上四叉树。 105

四头肌确实长在树上。	106
在四边形之外思考！	108
简单的基础知识.....再次。	109
宣传宣传。	114
像你以前从未剔除过的那样剔除.....再一次。	115
拥抱四叉树,爱四叉树,做四叉树。	116
实施基础。	116
使事情复杂一点。	123
加快速度。	125
概括。	125
参考。	126

第 7 章无论你在哪里

可以漫游。

漫游算法。	128
理论。	128
ROAM 算法的改进。	133
西莫的变化。	133
漫游 2.0。	137
第 1 步:实施基础知识。	137
第 2 步:添加平截头体剔除。	144
第 3 步:添加主干数据结构。	149
第 4 步:添加拆分/合并优先级队列。	156
概括。	163
参考。	164

第8章总结:特别

效果等等。 165

一切尽在水中。	166
让水流动,第 1 部分	166
让水流动,第 2 部分	170
基于基元的环境 101	174
在天空盒之外思考。	175
住在穹顶之下。	178
相机-地形碰撞检测和简单响应。	187
迷失在迷雾中。	189
基于距离的雾。	189
基于顶点的雾。	191
粒子引擎及其户外应用。	192
粒子引擎:基础。	192
将粒子带到一个新的维度。	197
添加数据插值。	199
将粒子引擎应用于户外场景。	201
概括。	202
结语。	202
参考。	203

附录CD 上的内容。 205

图形用户界面。	206
系统要求。	206
安装。	207
结构。	207

指数 。 209

来信
丛书编辑

迟早它必须发生。必须有人制作游戏
你实际上可以去外面做除了跑步以外的事情
在室内周围。不要误会我的意思;黑暗的时代,终极的邪恶,以及每个人周围的
成群结队的可怕生物
角落很有趣,但我们都需要一些光!专注于 3D 地形
编程就是这样。

每个人(即使您还没有创建自己的第一人称射击游戏)
3D 引擎)至少有一个想法,即二进制空间分区,八叉树,
门户网站和其他类似的技术通常用于这些
室内环境类型。然而,周围的神秘
地形编程和数千、数十万甚至数百万多边形的大规模户外渲染仍然是一个保存完
好的技巧。当然,如果您阅读 de Boer 的著作,诀窍就在那里

geomipmapping 算法,Rottger 的四叉树算法,或
Duchaineau 的 ROAM 算法,但白皮书中有这个词
对它们的抽象对我没有多大吸引力。我需要一点东西
更脚踏实地,实际上在现实世界中有效,并且
我可以使用的代码。这就是这本书的动机。

Focus On 3D Terrain Programming 是第一本书
100% 专注于地形编程,这是第一本书
这让几乎任何人都可以理解它。
而且书本小巧可爱,如果你愿意,可以购买上千本,打造真实的 3D 地形!不过说真的,
这本书太棒了。作者 Trent Polack 毫无疑问
实施了无数地形引擎和他的知识,
失败和成功将为您节省大量时间。

本书一开始就假设您对地形编程一无所知,除了将涉及一台计算机和

你可能需要向量<BG>。除此之外,
特伦特首先概述了地形规划和
然后你开始使用 OpenGL 作为你的第一个程序
选择的 API。如果您是 Direct3D 或软件 3D,请不要担心
家伙; OpenGL 就像 3D 的 C/C++。很明显是怎么回事
再加上它让所有的 Linux 人都开心,当然,这让 vi

销售强劲。无论如何，在介绍完之后，本书会立即进入各种流行的地形算法，从蛮力网格划分到 geomipmapping、四叉树和 ROAM。我会让你弄清楚缩写！

此外，纹理映射技术、照明和渲染地形的所有其他方面都通过工作演示进行了说明（并且有很多演示）。本书以水、网格动画、粒子系统（如雨）、雾化和广告牌等特殊效果结束。

我知道我经常这么说，但这是我最喜欢的书之一。最酷的是，您基本上对地形编程一无所知，然后在一个周末就什么都知道了！

另外，Trent 的代码是我见过的最好的代码。当文本中的概念难以理解时，您可以轻松地遵循他的逻辑。这是编程书籍的关键，因为许多作者没有他们需要解释的页面，或者它只是一个难以讨论的主题。因此，您只剩下代码来弥补差距，但如果代码是“hackeresk”，那么几乎不可能遵循。Trent 已经不遗余力地编写干净的代码，经常发表评论，并使用合理有效的编程结构，这些结构是最佳的，但并非不可能理解。

总之，Premier Game Development Series 再次凭借 Focus On 3D Terrain Programming 创造了另一个第一。我敢打赌，互联网将充斥着基于地形的 3D 游戏，而不是通常的在黑暗中跑来跑去的游戏，这将是一个不错的变化。我强烈推荐这本书给任何对户外 3D 渲染感兴趣并希望通过实验和学习压缩和过滤到这本书中的小宝石中的内容来节省大约 3 到 6 个月的时间。

真挚地，



安德烈·拉莫特

Premier 游戏开发系列的系列编辑器

介绍

欢迎关注 3D 地形编程 ! 这本书将带你从一个完全没有地形编程知识的新手程序员到一个可以实现一些最复杂算法的知识渊博的地形程序员。这本书到底是关于什么的 ? 好吧 , 我给你。

Focus On 3D Terrain Programming 是您对 3D 地形编程的回答。它全面涵盖了最流行的算法 (甚至是一种全新的算法) , 并以一种轻松有趣的阅读方式讨论了它们的所有概念。对于那些喜欢从视觉上看解释而不是试图从文本中理解它的人来说 , 所有的解释都是数字化的 , 并且解释还伴随着几个演示 , 所有这些都可以在本书的配套 CD 上找到。

准备好进入地形编程的丛林 ; 一旦你进去 , 即使是最大的激浪罐也不能把你拉出来。这本书的速度很快 , 但是任何有 C/C++ 经验的程序员只要对基本的 3D 理论有一点了解 , 就不会在理解上有困难。

无论讨论变得多么复杂 , 总会有乐趣 , 无论是实现一个很酷的新功能还是一个蹩脚的笑话。所以 , 事不宜迟 , 让我们开始旅程吧 !

第一部分

地形介绍

编程

1 进入之旅
伟大的户外活动

2 地形 101

3 纹理地形

4 光照地形

第1章

这

旅行

进入

伟大的

户外

欢迎来到我将成为您期待的精彩世界充满乐趣的地形书的向导，并且我们将一起渲染最高的山脉，最低的山谷，以及甚至可能是一片草叶。无论如何，在这本书中，您将学到关于地形编程及其在游戏开发中的应用的所有很酷的东西。所以，带上你的必需品

(你知道，音乐、咖啡因、袜子和你喜欢的小泰迪熊藏在你房间的深处)因为我们要得到开始了！

“地形？不，谢谢，我已经吃过了。”

我知道你要问的第一个问题是：“什么是大雨？”好吧，我会马上回答这个问题。地形是陆地：落基山脉，草地平原，连绵起伏的丘陵，共同组成一个美丽的风景。地形渲染领域关注如何实时渲染所有这些壮丽的自然特征。后你已经确定了地形，你需要弄清楚如何渲染自然的其他特征，例如水、云、太阳、雾和其他有趣的东西。

读完本书，你将完全理解如何创建一个非常逼真的户外场景，非常详细和高效。让我们对一些一般的地形信息做一个遍历，开始与地形的一般（非游戏开发）应用程序。

一般应用

在我们讨论 terrain 的超酷游戏开发应用程序之前，让我们先介绍一下它的其他一些应用程序。我发现了很多这样的 Virtual Terrain Project1 (<http://www.vterrain.org>) 上的信息，这是一个很好的网站，可以找到有关主题的一般信息

“地形?不,谢谢,我已经吃过了。”

5

地形及其所有应用。地形的一些应用
如下面所述:

- 虚拟旅游 (旅游规划)
- 天气和环境拓扑的可视化
- 房地产演练 ■军事用途,例如飞
行模拟器中的地形 (用于训练
目的)

这些只是地形的众多用途中的一小部分。如您所见,地形可视化和渲染是一个重
要的研究领域

几个原因。真正让地形渲染成为一个有用的工具

应用多,必须够详细,够快

以达到平滑的帧速率。(缓慢的应用完全扰乱了任何地形场景的真实感,真实感
是至高无上的

重要性。)这里提供的信息只是冰山一角;如果你对地形感兴趣而不是游戏开发,

查看前面提到的精彩网站。

地形和游戏开发

3D 地形在游戏开发中有着巨大的应用,尤其是
所有这些漂亮的、新的连续细节级别 (CLOD) 的出现
算法。(CLOD 算法的定义在后面解释

第 5 章,“为 CLOD 受损者进行地理映射。”)3D 游戏
以前受到基于户外游戏的巨大图形范围的影响。他们倾向于在小室内进行

房间和狭窄的走廊。(这在第一人称射击游戏类型中尤为常见。)

在过去的几年里,作为游戏玩家,我们看到了一系列跨越各种类型的出色户外游戏:
策略、动作和
第一人称射击游戏。黑白棋等游戏(见图 1.1)
和 Starsiege:部落是户外游戏的两个主要例子
广泛使用地形。

在过去几年发布的所有户外游戏中,只有一款
特别是可以对游戏中 3D 地形的普遍流行负责,并将其作为一般主题:
Treadmarks。



图 1.1 Black and White Studio 的黑白截图。

足迹

Seamus McNally 的Treadmarks于 2000 年 1 月发布,彻底改变了人们对游戏和其他应用程序中地形引擎的看法。该游戏如图 1.2 所示,基于坦克战和在基于 ROAM 的地形景观中竞速(其细节将在第 7 章“无论你在哪里漫游”中讨论),并涉及大量大爆炸。游戏最好的部分是每一次射击都会影响风景。例如,普通的炮弹会在地形上形成一个小洞,而较大的武器有可能会形成一个大陨石坑。

即使现在,这款游戏已经推出三年了,它仍然是任何专业制作游戏中最令人印象深刻的地形展示。

这主要是由于 McNally 实施了 ROAM 算法,该算法显示了一些新的想法和算法的变化,使其更适用于快节奏的图形应用程序,如Treadmarks或任何其他游戏。

“地形?不,谢谢,我已经吃过了。”

7



图 1.2 Seamus McNally 的Treadmarks,一款坦克格斗游戏,彻底改变了人们在游戏开发中查看地形的方式。

不幸的是,Seamus McNally 输掉了与霍奇金淋巴瘤并于 2000 年 3 月 30 日去世,享年 21 岁。虽然我不认识 Seamus 或他的家人(制作组 Treadmarks),我要感谢他的不可思议的想法和对地形可视化的思考,并希望他找到了平静。在 GameDev.net 上为他创建了一个纪念碑

(<http://www.gamedev.net/community/memorial/seumas>)。

因为Treadmarks对于地形编程世界来说是一个巨大的里程碑,所以我能够包含一个演示(来自 <http://www.treadmarks.com>)游戏随书附赠的 CD (Demos/TM_16_Demo.exe)。我强烈建议你如果可能的话,现在就去看看。游戏有一些很棒的地形效果,这是对我们将来要讨论的内容的一个很好的介绍贯穿本书。这只是一个如此令人上瘾的游戏!

演示构建变得简单！

本书的演示分为三组：主要章节演示、主要章节演示的替代版本以及各种程序员自愿包含在本书中的随机演示。所有这些以及更多内容都可以在本书出色的随附 CD 中找到。我现在将通过主要和备用书籍演示的编译说明。贡献的演示将不包括在内；它们将作为一个项目留给你去弄清楚。

主要演示

主要的演示是本书每一章的“官方”随附演示，由您真正编码。这些演示使用 OpenGL 进行渲染 API 和自定义 Windows 代码，因此您只能在 Windows 操作系统上运行它们。主要演示也使用 Microsoft Visual C++ 6.0 用 C++ 编写。

笔记

需要注意的是，尽管本书附带的演示坚持特定的 API，但实际的文本是 API 和操作系统无关的。无论你使用 OpenGL、Direct3D 还是任何其他 API，你都能理解本书的内容。

内容。

每章的代码分为两部分：演示代码和基础代码。

演示代码是将本书的所有理论和内容实现到演示中的地方，基本代码包含应用程序初始化、相机程序、数学运算等。

所有内容都放入名为 VC+
+(Microsoft Visual C+
+) 的工作区

demoXX_YY.dsw，其中 XX 是章节编号，YY 是当前章节的演示编号。当您在 VC++ 中打开工作区时，您可以构建应用程序，它应该可以顺利编译。让我们逐步实现 demo1_1，您可以在随附 CD 的 Code\Chapter 1\demo1_1 下找到它。

首先，打开 Microsoft Visual C++ 和 demo1_1.dsw（文件，打开工作区）。完成此操作后，项目工具栏应如图 1.3 所示。

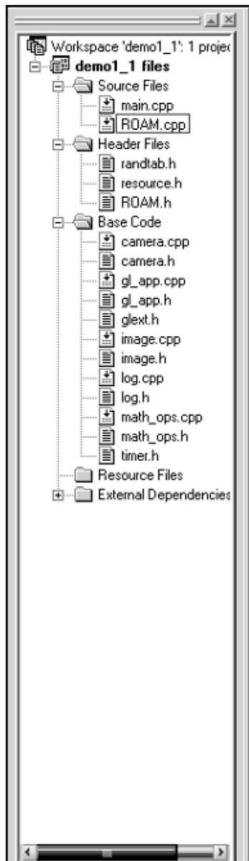


图 1.3 Microsoft Visual C++ 6.0 中 demo1_1 的项目工具栏。

从那里,您可以简单地构建 (Build, Build demo1_1.exe)

演示,然后执行EXE。演示几乎是一模一样的复制品

demo7_1 (第 7 章的第一个演示) ,所以你可以期待在本书后面看到一些类似的东西。不过现在,可以做一个快速的解释。该演示展示了 ROAM 的简单镶嵌

算法看起来像。既然你已经有了那个小预告,那就等着吧

直到你到达相当大的第 7 章!如果你正确构建了 demo1_1,它应该如图 1.4 所示。

还要看看控件

用于表 1.1 中的演示。

表 1.1 demo1_1 的控件

钥匙	功能
逃生/Q	退出程序
向上箭头	前进
向下箭头	往后退
右箭头	向右扫射
左箭头	左扫射

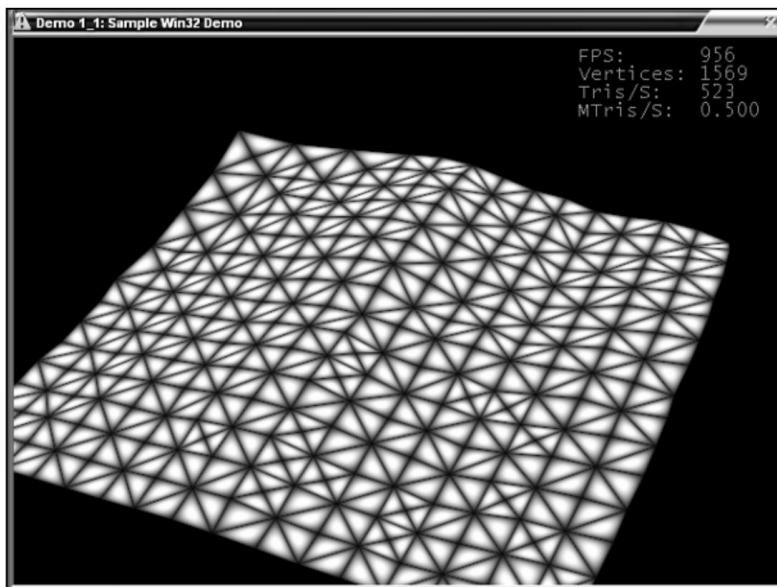


图 1.4 demo1_1 的截图。

这不是漂亮的截图吗?我认同。无论如何,这是一个跛脚

我们将在本书中完成的示例,但它确实有效

一个小预告!

本书一目了然

这本书将是关于地形、地形、地形和*喘气*甚至更多地形!我们将涵盖从分形高度图生成到三种不同的 CLOD 算法的所有内容。我们将以一个非常大的章节结束,介绍可用于增加任何 3D 地形场景的真实感和细节的特效。这些影响包括

例如雾,云渲染、镜头光晕和其他各种技巧,
技巧和效果。因此,事不宜迟,让我们进入摘要。
我们将逐章阅读这本书,而不是逐章阅读。

第一部分 :简介

地形编程

本部分可帮助您轻松进行地形编程。第 2 章,“地形 101” ,
讨论高度图操作,例如加载和保存高度图,以及
然后继续讨论生成高度图。(高度图是
定义地形引擎中每个顶点高度的 2D 图像。)
高度图生成部分真的很整洁,因为它教你
如何用很少的工作创建看起来很酷的高度图。

第 3 章 “纹理地形”从在地形网格上拉伸单个纹理开始。从那里开始,
本章继续
讨论程序纹理生成,它比拉伸单个草或泥土纹理产生更好的外观效果。然后,
本章将所谓的细节纹理添加到地形,

极大地改善了景观的视觉外观。

第 4 章 “光照地形”是第一部分的最后一章。它涵盖了简单的地形照明技术。从
非常简单的基于高度的地形光照,然后本章继续介绍光照贴图地形。本
章以
令人难以置信的斜坡照明技术,提供了很棒的
用最少的代码照明。

第二部分:高级

地形编程

第 5 章,“为 CLOUD 受损者进行地理映射”,第 6 章,“攀爬四叉树”和第 7 节“无论你在哪里漫游”处理基于 CLOUD 的地形算法。一句话,CLOUD 算法是一个

动态多边形网格,“给予”额外的三角形区域

需要更多细节。这是对此事的简单解释,但在我们深入阅读本书之前它会起作用。以下是

涵盖的算法:

- Willem H. de Boer 的地理映射算法
- Stefan Roettger 的四叉树算法
- Mark Duchaineau 的 ROAM 算法

这些究竟是什么将及时解释,但简要说明:它们是真的很酷!每个都以自己的方式很棒,并且每个都与其他完全不同,这为以后的编码体验带来了相当多的变化。第 5 章、第 6 章和第 7 章将让你大吃一惊

通读。然后,我们将以各种特别的方式结束这本书
为第 8 章“总结:特殊效果等”中前面提到的实现“增添趣味”的效果和技巧。在

在那一章,我们将介绍一些很酷的效果,例如雾、变形、
和其他“环境”影响。

演示

我为讨论的每个主要主题编写了一个演示程序
贯穿本书。当我们继续阅读本书时,您必须记住我提供的演示只是

用作您自己实施的垫脚石。我的实现是为您提供良好的教学指南

演示;不要只是将演示代码复制并粘贴到您自己的
项目。提供的 demo 没有高度优化,不提供
最佳细节,不要实现所有的花里胡哨
我们将讨论的各种技术。

因为我是很好的人,所以我决定帮助你一点。地形是
一个动态的问题:有朝一日可行的技术可能需要
彻底大修,改天有用。因此,我有
将我的大部分网站 (<http://trent.codershq.com/>) 用于地形

总结13

研究和实施,我将保持我在 3D 地形编程领域的进展的恒定数据库。我将尽可能地开发出最详细和最快速的实现,并且我会持续记录我所做的开发。如果本书 CD 上提供的演示对您来说还不够,请务必查看我的网站以获取一系列演示和信息,它们将成为本书的宝贵伴侣。

概括

本章介绍了地形及其应用的基础知识。它还研究了如何编译和执行两种不同类型的演示,并提供了整本书的概述。准备好:您在地形渲染的美妙世界中的旅程即将开始!

第2章

地形101

好吧,这就是你第一章!本章将涵盖所有方面

在开始使用纹理/照明技术以及各种“硬核”地形算法之前,您需要了解的地形渲染。在本章中,您将学习以下关键概念:

- 什么是高度图、如何创建以及如何加载它们 ■如何使用蛮力算法渲染地形 ■
如何使用两种算法生成分形地形:断层形成和中点位移

所以,事不宜迟,让我们开始吧!

高度图

想象一下,您有一个沿 X 轴和 Z 轴延伸的规则多边形网格。如果你不知道我在说什么,图 2.1 可能会让你记忆犹新。

现在这是一个非常无聊的图像!我们究竟要如何让它变得更加好、更接近地形?答案是使用高度图。在我们的例子中,高度图是一系列 unsigned char 变量(它让我们的值在 0-255 的范围内,恰好是灰度图像中灰度的数量),我们将在运行时或在绘图程序中。这个高度图定义了我们地形的高度值,所以如果我们的网格沿着 X 轴和 Z 轴,高度图定义了将网格扩展到 Y 轴的值。举个简单的例子,查看图 2.2 中的高度图。在我们加载它并将其应用到我们的地形后,图 2.1 中的网格将转变为我们在图 2.3 中看到的美丽地形(尽管它非常缺乏颜色和照明)。

诚然,图 2.3 中的地形在没有很酷的纹理或光照的情况下看起来很无聊,但我们需要从某个地方开始!正如我刚刚解释的那样,高度图使我们能够将无聊的顶点网格塑造成壮丽的景观。问题是,究竟是什么

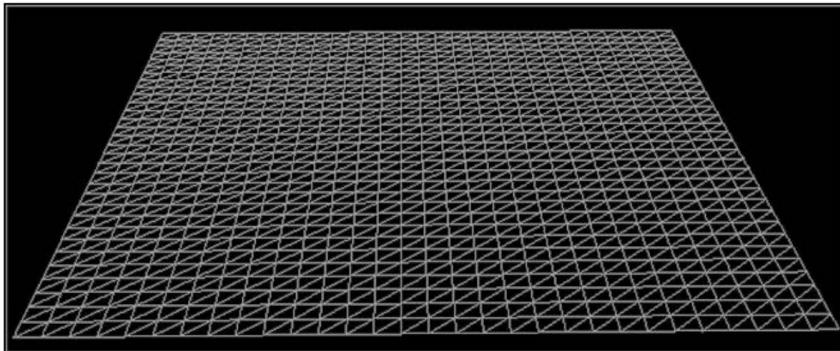


图 2.1 具有未定义高度值的顶点网格。

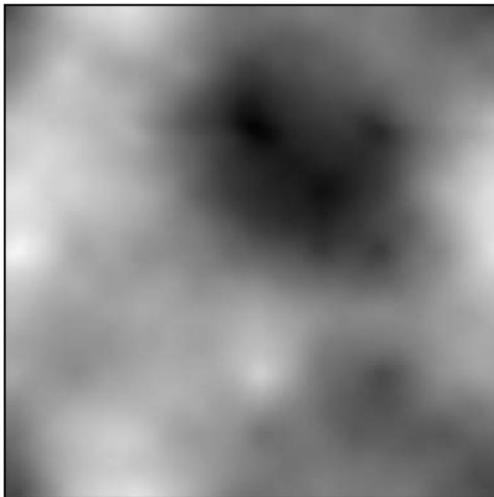


图 2.2 用于创建图 2.3 的 128×128 高度图。

高度图?通常,高度图是灰度图像,其中
每个像素代表一个高度值。(在我们的例子中,高度范围
从 0 到 255,灰度图像中灰度的数量。)暗
颜色代表较低的海拔,较浅的颜色代表
海拔较高。参见图 2.2 和 2.3;注意 3D 地形(图 2.3)如何与图中的高度图
完全对应
2.2,从山峰到山谷,甚至颜色的一切?这就是我们希望我们的高度图做
的事情:给我们力量
“塑造”一个顶点网格以创建我们想要的地形。

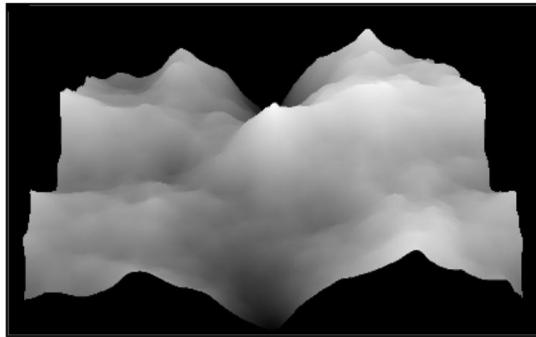


图 2.3 使用图 2.2 中的高度图创建的蛮力地形图
像。

在我们的例子中,高度图的文件格式将是
RAW 格式。(虽然大多数演示动态创建高度图,但我包括使用 RAW 保存/加载高度图的选项

格式。)我选择这种格式只是因为它非常简单
利用。此外,由于 RAW 格式只包含纯数据,因此它是
更容易加载高度图。(我们也在以灰度加载
RAW 图像,这使事情变得更容易。)在我们加载 RAW 之前
形象,我们需要做几件事。首先,我们需要创建一个
可以表示高度图的简单数据结构。我们需要的
因为这个结构是 unsigned char 变量的缓冲区(我们需要
能够动态分配内存)和一个变量来跟踪
高度图的大小。够简单吧?好吧,这里是:

```
结构 HEIGHT_DATA
{
    无符号字符* m_pucData; //高度数据
    诠释 m_iSize;           //高度大小 (2的幂)
};
```

创建一个 基本地形类

我们需要创建一个基类,从中我们所有的特定地形
引擎(蛮力,geomipmapping 等)将被派生。

我们不希望用户实际创建此类的实例;我们只是希望这个类成为我们特定的共同父级

创建基础地形类

我们稍后将开发的实现。请参阅图 2.4 以获得视觉效果
我的想法。

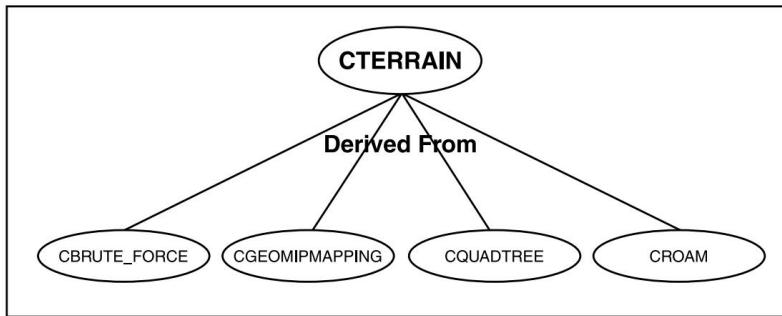


图 2.4 CTERRAIN与四种地形实现的关系。

笔记

CTERRAIN类就是我们的 C++
瘾君子喜欢将其称为摘要
类。抽象类是一个类
功能作为一个通用接口
它所有的孩子。这样想：
一位母亲是一头红头发,但性格很无聊。虽然她的所有
孩子继承了母亲的
红头发,每个人都有独特的个性,非常有趣。

这同样适用于摘要
班级;虽然抽象类是
“无聊”本身,它的特点继续
给它的孩子,以及那些孩子
可以定义更多“令人兴奋”的行为
为自己。

到目前为止,我们所需要的一切
我们的基类是三个变量:一个实
例

SHEIGHT_DATA,一个高度
缩放变量 (它将让
我们动态缩放
我们地形的高度),
和一个大小变量 (其中
应该与 size 成员相同)

SHEIGHT_DATA)。据,直到...为止
函数去,我们需要
一些高度图操作函数和一个设置高度的
函数

缩放变量。这里是
我想出了什么:

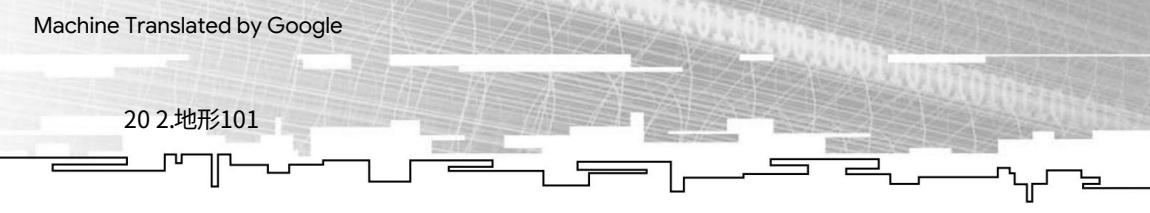
CTERRAIN 类

{

受保护:

SHEIGHT_DATA m_heightData; //高度数据

20 2.地形101



```

    浮动 m_fHeightScale;           //缩放变量

    上市:
        诠释 m_iSize;           //必须是2的幂

    虚拟无效渲染 (无效)= 0;

    bool LoadHeightMap(char* szFilename, int iSize);
    bool SaveHeightMap(char* szFilename);
    布尔卸载高度图 (无效) ;

    //

    // 姓名:             SetHeightScale - 公共
    // 描述:             设置高度缩放变量
    // 参数:             -fScale:缩放地形多少
    // 返回值:无

    //

    内联 void SetHeightScale( 浮动 fScale )
    {
        m_fHeightScale= fScale;
    }

    //

    // 姓名:             SetHeightAtPoint - 公共
    // 描述:             设置给定点的真实高度值
    // 参数:             -ucHeight:点的新高度值
    //                   -iX, iZ: 要检索的高度值
    // 返回值:无

    //

    内联无效 SetHeightAtPoint (无符号字符 ucHeight,
                                你iX,你iZ)
    {
        m_heightData.m_pucData[(iZ*m_iSize)+iX]= ucHeight;
    }

    //

    // 姓名:             GetTrueHeightAtPoint - 公共
    // 描述: //          获取真实高度的函数
    //       某点的值 (0-255)
    // 参数:             -iX, iZ: 要检索的高度值
    // 返回值:一个无符号字符值:真实高度
    // 给定点

```

加载和卸载高度图

21

```

//  

内联 unsigned char GetTrueHeightAtPoint( int iX, int iZ )  

{ 返回 ( m_heightData.m_pucData[ ( iZ*m_iSize )+iX ] ); }  

//  

// 姓名:           GetScaledHeightAtPoint - 公共  

// 描述:检索给定点的缩放高度  

// 参数: -iX, iZ: 要检索的高度值  

// 返回值:一个浮点值,给定的缩放高度  

//           观点  

//  

内联浮点 GetScaledHeightAtPoint( int iX, int iZ )  

{ 返回 ( (m_heightData.m_pucData[(iZ*m_iSize)+iX]  

           )*m_fHeightScale ); }  

地形 (无效)  

{ }  

~CTERRAIN (无效)  

{ }  

};

};

如果我自己这么说,也不会太破旧!那是我们的“父”地形  

班级!我们开发的所有其他实现都派生自这个类。  

我在课堂上放了很多高度图操作函数只是为了  

让我们和用户都更轻松。我包括了两个高度  

检索功能是有原因的。而我们,作为开发人员,将使用  

“真实”功能最常使用,用户将使用“缩放”  

最常执行碰撞检测的函数(我们将  

在第8章,“总结:特殊效果等”中做)。

```

加载和卸载高度图

我已经谈论这两个例程有一段时间了,它是
是时候我们终于直接潜入它们了。这些例程是
很简单,所以不要让它们变得比应有的更难。我们是
只是做一些简单的C风格文件I/O。

22.2.地形101

笔记

我倾向于坚持使用 C 风格的 I/O,因为它比 C++ 风格的 I/O 更容易阅读。如果

您真的是一个真正的 C++ 迷,并且绝对讨厌 C 的做事方式,那么请随意将例程更改为“真正的”

C++。另一方面,我真的很喜欢 C++ 风格的内存操作,所以,如果你是一个“真正的”C 迷,那就改变它们吧。

我们需要谈谈如何加载、保存和卸载高度图。最好的起点是加载例程,因为没有加载就无法卸载。

我们需要函数的两个参数:文件
地图的名称和大小。在函数内部,我们要创建一个FILE实例,以便我们可以加载请求的高度图。然后我们要确保该类的高度图

实例尚未加载信息;如果是,那么我们需要调用卸载程序并继续我们的业务。这是我们刚刚讨论的代码:

```
bool CTERRAIN::LoadHeightMap(char* szFilename, int iSize) {
```

```
    文件* pFile;
```

```
    //检查数据是否已经设置
```

```
    如果 (m_heightData.m_pucData)
```

```
        卸载高度图();
```

接下来,我们需要打开文件并在高度图实例的数据缓冲区 (m_heightData.m_pucData) 中分配内存。我们需要确保内存分配正确,并且没有出现可怕的错误。

```
//为我们的高度数据分配内存 m_heightData.m_pucData= new
```

```
unsigned char [iSize*iSize];
```

```
//检查内存是否分配成功 if( m_heightData.m_pucData==NULL )
```

```
{
```

```
    //内存无法分配 //这里有严重错误 printf( "Could not  
allocate memory for% s\n" ,szFilename );
```

加载和卸载高度图23

```
    返回假;
}
```

对于我们加载过程的倒数第二个步骤,我们将加载实际数据并将其放置在高度图实例的数据缓冲区中。

然后我们要关闭文件,设置一些类的成员变量,并打印一条成功消息。

```
//将高度图读入上下文 fread( m_heightData.m_pucData,
1, iSize*iSize, pFile );

//关闭文件
fclose(pFile);

//设置尺寸数据
m_heightData.m_iSize=iSize; m_iSize
= m_heightData.m_iSize;

//雅虎!高度图已成功加载! printf( “加载的 %s\n” ,szFilename );

返回真;
}
```

笔记

高度图保存例程与加载例程几乎相同。
基本上,我们只需要用 fwrite 替换 fread。
这就是它的全部内容!

这就是加载
常规。让我们继续

在我失去你的注意力之
前的卸载程序。卸货过程
很简单。我们只需要检查内存
是否真的被分配了,如果有,
我们需要删除它。

布尔 CTERRAIN::UnloadHeightMap(无效) {

```
//检查数据是否已经设置
如果 (m_heightData.m_pucData){

//删除数据
删除[] m_heightData.m_pucData;
```

24.2. 地形101

```
//重置地图尺寸,m_heightData.m_iSize=0;
}

//高度图已被卸载 printf(“成功卸载高度图\n”);

返回真;
}
```

我确实不需要检查数据缓冲区是否为NULL指针（`delete`内部检查指针是否为NULL），所以我的检查有点多余。然而，检查是我已经养成的一种习惯，所以我将在本书中一直这样做。只需知道您可以在不先检查 NULL 指针的情况下调用 `delete`。现在是时候向您展示一种简单的渲染我们刚才讨论的内容的方法了。

物质的蛮力

使用蛮力算法渲染地形非常简单，并且它提供了尽可能多的细节。不幸的是，它是本书介绍的所有算法中最慢的。基本上，如果你有一个 64×64 像素的高度图，那么当使用蛮力渲染时，地形由 64×64 个顶点组成，以规则的重复模式（见图 2.5）。

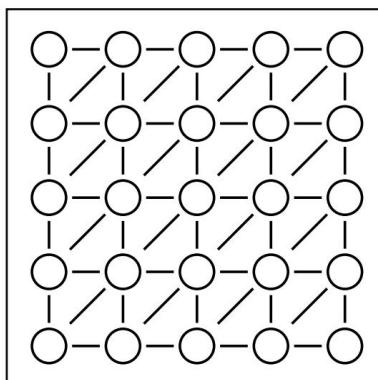


图 2.5 一个 5×5 的蛮力地形顶点补丁。

物质的蛮力25

如果您没有立即识别它，我们会将每一行顶点渲染为三角形条带，因为这是渲染顶点的最合乎逻辑的方式。您不会完全希望将它们渲染为单独的三角形或三角形扇形，其结构类似于图 2.5 中呈现的结构，对吗？

对于本章的演示，我尽量保持简单。顶点的颜色将基于其高度，因此所有顶点都是灰色阴影。这就是使用蛮力渲染地形的全部内容。这是一个使用 OpenGL 的快速片段，展示了我们将如何渲染地形：

```

无效 CBRUTE_FORCE:: 渲染 (无效)
{
    无符号字符 ucColor; 诠释 iZ;

    你是 X;

    //遍历地形的 Z 轴 for( iZ=0; iZ<m_iSize-1; iZ++ )

    {
        //开始一个新的三角形带
        glBegin( GL_TRIANGLE_STRIP );

        //循环穿过地形的 X 轴 //这是构造三角形带的地方

        for(iX=0;iX<m_iSize-1;iX++)
        {
            //使用基于高度的着色。 (高点是//亮,低点是暗。) ucColor=
            GetTrueHeightAtPoint( iX, iZ );

            //用OpenGL设置颜色，并渲染点 glColor3ub( ucColor, ucColor, ucColor );
            glVertex3f(iX, GetScaledHeightAtPoint(iX, iZ), iZ);

            //使用基于高度的着色。 (高点是//亮,低点是暗。) ucColor=
            GetTrueHeightAtPoint( iX, iZ+1 );
        }
    }
}

```

26 2.地形101

```
//用OpenGL设置颜色,并渲染点 glColor3ub( ucColor, ucColor,
ucColor ); glVertex3f(iX,
GetScaledHeightAtPoint(iX, iZ+1), iZ+1);

}

//结束三角形带 glEnd( );

}

}

现在是您创建的第一个实际演示的时候了!查看 CD 上的 demo2_1。转到
Code\Chapter 2\demo2_1,在 Microsoft Visual C++ 中打开工作区,开始享受
乐趣吧!该演示展示了我们刚刚讨论的所有内容。图 2.6 显示了演示的屏幕截图,表 2.1
提供了演示控件的描述。要移动您的视点,只需按住鼠标左键或右键并拖动鼠标。
```



图 2.6 来自 demo2_1 的屏幕截图。

呜呼!现在,我刚才说过,我们将动态创建大部分高度图。你可能会问自己,“我该怎么做?”好吧,我很高兴你问。(即使你没有,我仍然会解释它!)现在我们将学习如何使用两种分形地形生成技术来生成高度图。做好准备!

分形地形生成27

表 2.1 演示 2_1 的控件

钥匙	功能
逃生/Q	退出程序
向上箭头	前进
向下箭头	往后退
右箭头	向右扫射
左箭头	左扫射
在	以线框模式渲染
小号	以实体/填充模式渲染
+ / -	增加/减少鼠标灵敏度
] / [增加/减少移动灵敏度

分形地形生成

分形地形生成是算法生成地形的过程,尽管在我们的例子中,我们只是生成一个高度图

用作我们地形的“蓝图”。我们将通过

这里有两种算法,第一种是断层形成,第二种是中点位移。我们将使用故障

形成算法通过大部分书,因为它没有

对生成的必须使用哪些维度进行限制

高度图,而中点位移要求尺寸是 2 的幂。(尺寸也必须相等,所以

虽然可以生成 1024×1024 的高度图,但不能

生成一个 512×1024 的高度图。)所以,事不宜迟,让我们

开始使用分形地形生成算法!

断层形成

一种分形地形生成方法称为断层形成。

断层形成是在地形中产生“断层”的过程;为了

28.2.地形101

大多数情况下,它会产生相当平滑的地形。基本上,我们所做的只是在空白高度字段中添加一条随机线,然后在其中一侧添加随机高度。如果你无法想象这个,或者你只是想确认你脑海中的图像(或者,如果你像我一样,你脑海中的声音 我很奇怪)是正确的,请参见图 2.7。

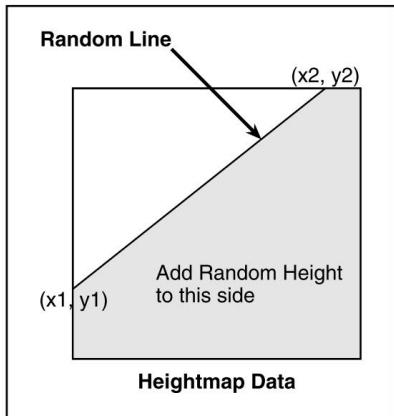


图 2.7 断层形成算法的第一步。

当然,这只是整个过程的第一步。在达到高级阶段之前,您还需要了解一些关于算法的知识。首先,我之前谈到的高度值需要随着每次迭代而减小。为什么,你可能会问?好吧,如果您在每次通过后不降低高度,您最终会得到如图 2.8 所示的高度图。请参阅图 2.9 了解高度图应该是什么样子的示例。

请注意,在图 2.8 中,亮/暗点如何没有韵律或原因;它们只是散布在各处。这对于混乱的地形来说很好,但我们想要创建平滑、连绵起伏的山丘。没有恐惧;解决这个问题相当简单。我们希望线性减小高度值而不使其降至零。为此,我们使用以下等式(来自 demo2_2) :

```
iHeight= iMaxDelta - (( iMaxDelta-iMinDelta )*iCurrentIteration  
)/iterations;
```

分形地形生成29

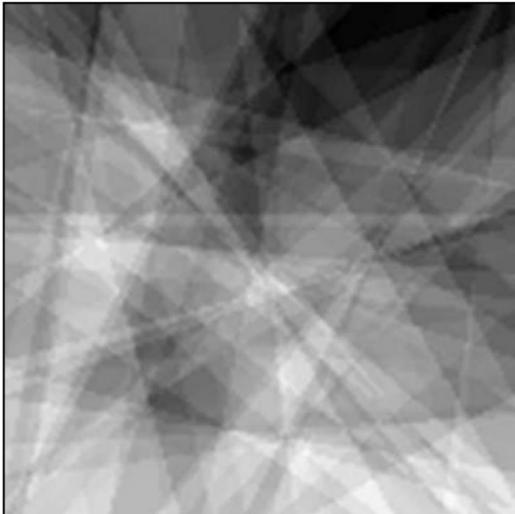


图 2.8 使用完全随机高度值生成的地图。

`iMinDelta`、`iMaxDelta`和`Iterations`都作为函数参数提供。`iMinDelta`和`iMaxDelta`代表最低和最高

（分别）形成新时所需的高度值
故障。我倾向于坚持使用`iMinDelta`的 0 值和 255 的值

对于`iMaxDelta`。`Iterations`,正如我之前所说,代表了
故障通过（高度图应该多少次不同
分开）。最后,但同样重要的是, `iCurrentIteration`表示当前迭代次数。

正如我之前所说,我们只想提升线路的一侧,我们
想要提高该线那一侧的每个点的高度值。

因此,我们将不得不遍历整个高度图的所有高度值。这一切都很容易实现;这只是

涉及一些简单的数学。我们有一个向量,它沿着我们的直线方向（由我们定义的两个随机点定义）

之前创建的）,其方向存储在(`iDirX1`,`iDirZ1`) 中。这
我们要创建的下一个向量是来自初始随机数的向量
点(`iRandX1`,`iRandZ1`)到循环中的当前点(x , z)。后
完成后,我们需要找到叉积的 Z 分量,
如果它大于零,那么我们需要提升电流
有问题的点。前面的所有解释都显示在下面
演示中的代码。

30 2.地形101

//iDirX1,iDirZ1是一个与直线同向的向量

iDirX1= iRandX2-iRandX1;

iDirZ1 = iRandZ2-iRandZ1;

对于 (x=0;x<m_iSize;x++)

{

for(z=0;z<m_iSize;z++)

{

//iDirX2, iDirZ2 是从 iRandX1, iRandZ1 到 //当前点 (在循环中)的向量。 iDirX2= x-
iRandX1;

iDirZ2 = z-iRandZ1;

//如果 (iDirX2*iDirZ1 - iDirX1*iDirZ2) 的结果是 “向上” //(高于 0),则将此点提高 iHeight

if((iDirX2*iDirZ1 - iDirX1*iDirZ2)>0)

fTempBuffer[(z*m_iSize)+x]=(float)iHeight;

}

}

笔记

当您查看这两个段和 demo2_2 中的故障形成和中点位移代码时,您可能会注意到我是如何创建一个浮点值的临时缓冲区fTempBuffer以将所有高度值放入其中。如果您还记得,不过,我谈到我们的高度图是一组无符号字符变量。为什么在这种情况下我要使用浮点变量?我这样做是因为该算法需要比我们正常的unsigned char高度缓冲区更高的精度。在我们创建并规范化整个高度图之后,我将所有信息从fTempBuffer 传输到CTERRAIN类中的高度缓冲区m_heightData。

查看图 2.9 以查看使用断层形成和变化形成的几个高度图

故障线数

迭代。

就我们而言,我们还没有完成这个算法!

如果您没有注意到,上图中的地图看起来是非地形的(新词)。我们需要在整个地图上传递一个侵蚀过滤器

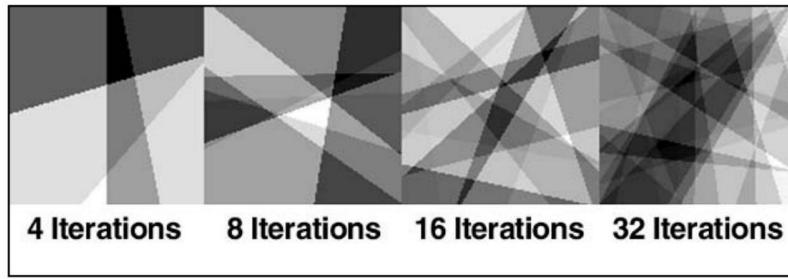


图 2.9 在几次“故障形成”通过后创建的高度图示例。

在我们形成一个新的断层以平滑我们拥有的值之后。这个

这个过程很像,如果不完全是,就像在你最喜欢的绘图程序中的图像上传递一个模糊过滤器。如果它有助于您理解以下解释,请这样想。

我们要做的是应用一个简单的 FIR 滤波器,正如 Jason Shankel 所建议的那样。¹该滤波器旨在模拟自然界中经常发生的地形侵蚀。(你见过自然界中的一系列山脉,看起来像图 2.9 中的高度图吗?)我们将从波段中获取数据,而不是一次过滤整个高度图。过滤函数如下所示:

```
void CTERRAIN::FilterHeightBand(float* fpBand, int iStride,
                                 int iCount, 浮动 fFilter)
{
    浮动 v=ucpBand[0]; int j=
    iStride;诠释我;

    //通过高度带并应用侵蚀过滤器
    for(i=0;i<iCount-1;i++)
    {
        ucpBand[j]=fFilter*v + (1-fFilter)*ucpBand[j];

        v = ucpBand[j]; j+= 步
        帧;
    }
}
```

32 2.地形101

此函数采用单个高度值带并逐个值遍历它们,iStride 指示循环中每次迭代将值推进多少。 iStride 还规定了前进的方向,因为我们将从上到下、从下到上、从左到右和从右到左过滤整个高度场。整个函数中最重要的一行是这一行:

```
ucpBand[j]= fFilter*v + ( 1-fFilter )*ucpBand[j];
```

这是模糊/侵蚀的线。 fFilter的不同值会影响模糊。 0.0f根本没有模糊，1.0f是非常强烈的模糊。

通常,我们希望值在0.3f到0.6f 的范围内,具体取决于您希望地形的平滑程度。现在,例如,假设我们的过滤器值为 0.25f,当前波段值为 0.9f。前面的等式如下所示:

```
ucpBand[j]= 0.25f*v + (1-0.25f)*0.9f;
```

在我们执行初始计算之后,前面的等式将简化为:

```
ucpBand[j]= 0.25f*v + 0.675f;
```

0.675f 是我们正在模糊的高度图像素的新值,但现在需要用它之前的像素进行插值。(我们将给该像素一个 0.87f 的值。)我们将 0.25f 模糊滤镜值应用于该像素,并将其添加到我们尝试计算的像素的未插值像素值中:

```
ucpBand[j]= 0.25f*0.87f + 0.675f;
```

进行最终计算,我们最终得到 0.8925f 的值。所以,你看,我们在这里真正做的就是将前一个像素与当前像素“混合”一点。查看图 2.10,看看 filter 看起来比我们之前讨论的每像素操作的规模大得多。

稍微玩一下 demo2_2。我为高度图操作创建了一个新的菜单区域,现在您可以动态创建新的高度图。如果你找到你喜欢的,只需选择 Save Current 选项,高度图就会保存到程序的目录中。

当您选择“断层形成”选项时,会打开一个对话框并提示您输入详细值。这个值是一个整数值,所以保持数字在 0-100 的范围内。现在,是时候享受一些中点置换的乐趣了!

分形地形生成33

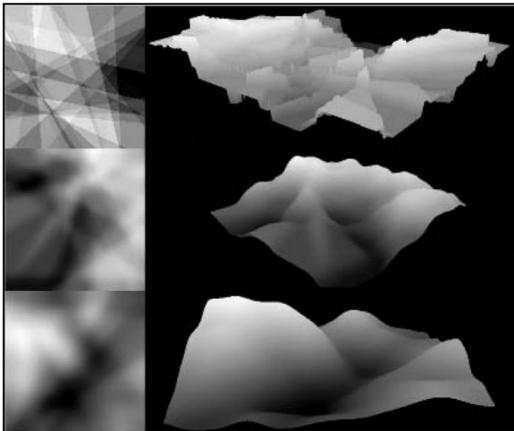


图 2.10 使用断层形成算法和侵蚀过滤器生成的高度图。
上图的滤镜值为 0.0f, 中图的滤镜值为 0.2f, 下图的滤镜
值为 0.4f。

中点位移断层的形成非常适合由一些小山丘组成的
漂亮小场景,但如果你想要比这更混乱的东西,比如山脉怎么办?好吧,别
再看了。中点位移2是您正在寻找的答案!该算法也称为等离子分形算法和菱
形正方形算法。然而,中点位移听起来很酷,它让读者(就是你)更好地了解
整个过程中实际发生的事情,所以我大部分时间都会坚持这个术语

笔记

需要注意的是,中点位移算法有一个小缺
点:该算法只能生成正方形的高度图,并且
尺寸必须是 2 的幂。这与断层形成算法不
同,在该算法中,您可以指定任何您想要的
尺寸。

时间。

本质上,我们在这个算法
中所做的就是取一条线的中
点并移动它!让我给你一个一
维的贯穿。如果我们有一条简
单的线,例如图 2.11 中的
AB,我们将取它的中点,在图
中表示为 C,然后移动它!

34 2.地形101

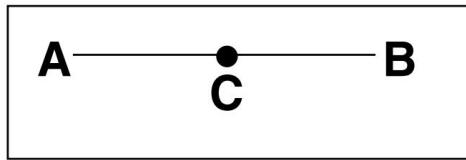


图 2.11一条简单的线,它是一维版本算法的第一阶段。

现在,我们将用一个高度值来移动那条线的中点,我们称之为fHeight (见图 2.12)。我们将使它与所讨论的线的长度相等,并将中点移动-fHeight/2到fHeight/2 的范围。(我们希望每次将线细分为两部分,并且希望在该范围内的某处置换线的高度。)

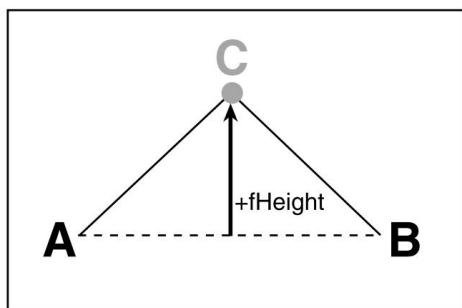


图 2.12图 2.11 经过一次位移后的线。

第一遍之后,我们需要减小fHeight的值以达到我们想要的粗糙度。为此,我们只需将fHeight乘以 $2-fRoughness$,其中fRoughness是一个常数,表示所需的地形粗糙度。用户将指定fRoughness 的值,因此您需要了解可以为其设置的各种值。

从技术上讲,该值可以是您心中想要的任何浮点值,但最好的结果是从0.25f到1.5f。查看图 2.13,了解不同粗糙度级别可以做什么的视觉指示。

如您所见,您为fRoughness传递的值极大地影响了高度图的外观。低于1.0f的值会产生混乱的地形, 1.0f的值会产生相当“平衡”的外观,而

分形地形生成35

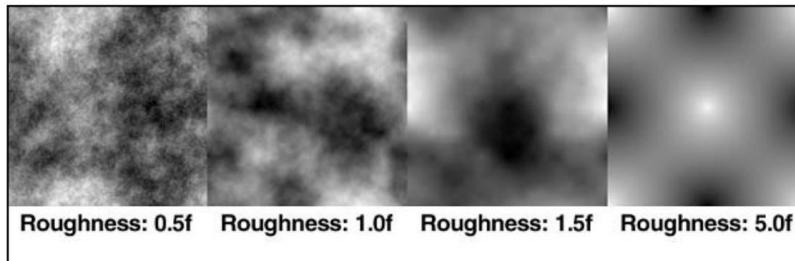


图 2.13 变化的值传递 fRoughness。

大于 1.0f 创建平滑的地形。现在,让我们将这个解释国家踢入第二维度。

当我们讨论要为 2D 解释更改什么时,请不断将 1D 解释留在您的脑海中,因为您刚刚为该单行学到的每个概念仍然适用。例外的是,我们现在必须计算四条不同线的中点,将它们平均,然后在正方形的中间加上高度值,而不是计算单条线的中点。图 2.14 显示了我们开始时的空白方块 (ABCD)。

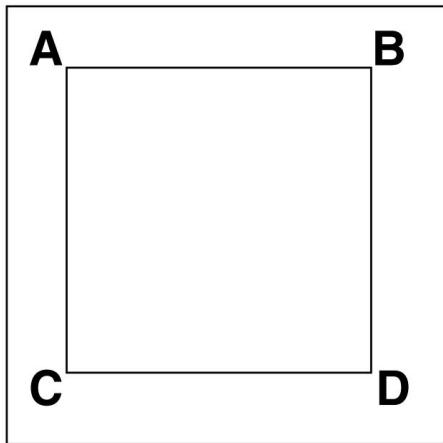


图 2.14 算法的 2D 版本的第一阶段。 (尚未发生位移。)

正如我刚才所说,我们必须计算所有四条线 (AB、BD、DC、CA) 的中点。结果点 E 应直接位于

36 2.地形101

广场中间。然后,我们通过取 A、B、C 和 D 的高度值的平均值来置换 E,然后我们在- $fHeight/2$ 到 $fHeight/2$ 的范围内添加一个随机值。这会产生如图 2.15 所示的图像。

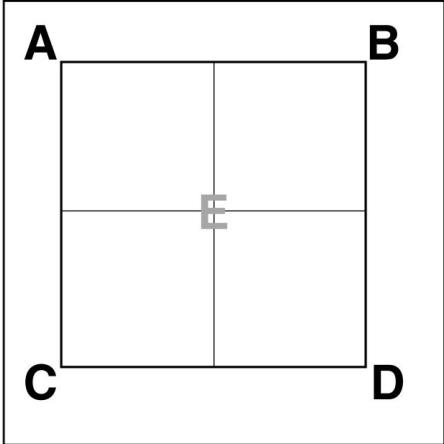


图 2.15 算法 2D 版本中的前半部分位移阶段。

那只是第一个位移阶段的前半部分。现在我们必须计算我们之前找到的每个中点的高度值。不过,这与我们之前所做的类似;我们只是平均周围顶点的高度值,并在 $fHeight/2$ 到 $fHeight/2$ 的范围内添加一个随机高度值。你最终会得到一个如图 2.16 所示的正方形。

然后,您求助于下一组矩形并执行相同的过程。但是,如果您了解 1D 说明,那么您肯定会理解 2D 说明和随附的代码 demo2_2,该代码位于 CD 的 Code\Chapter 2\demo2_2 下。

像往常一样,编译信息在演示目录中以文本文件的形式提供。去看看演示。控件与上次相同(请参阅表 2.1 进行提醒),但这次,当您单击 Detail 字段的 Midpoint Displacement 时,您希望值在 0 (非常混乱的地形)到 150 (简单的地形)的范围内,玩得开心!

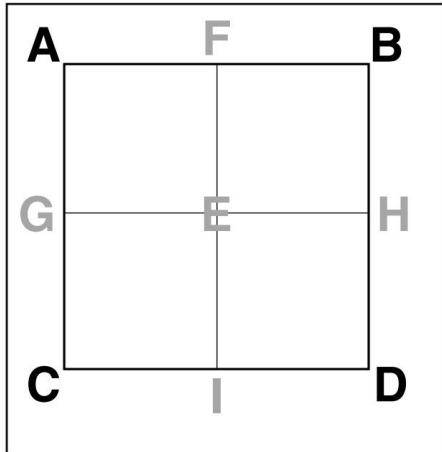


图 2.16 第一个位移阶段的最后一步。

概括

在本章中,您获得了地形编程的入门学位。您了解了有关高度图的所有知识:它们是什么、如何生成它们以及如何加载/卸载它们。然后,您学习了如何使用市场上最简单(也是最好看)的地形算法蛮力来渲染这些高度图。最后,您学习了两种以程序方式为地形生成高度图的方法。在接下来的两章中,我们将学习如何使用酷炫的纹理和照明技术来“增加”我们的地形。

参考

1 香克尔,杰森。“分形地形生成 断层形成。”
游戏编程宝石。马萨诸塞州罗克兰:查尔斯河媒体,2000. 499–502。

2 香克尔,杰森。“分形地形生成 中点位移。”游戏编程宝石。马萨诸塞州罗克兰:查尔斯河媒体,2000. 503–507。

第3章

纹理

地形

40 3. 纹理地形

现在您已经了解了制作简单地形的方法,你需要制作简单的地形贴图为那个无聊的网格添加细节。我将让这个关于纹理的讨论简单明了,这样我们就可以开始真正有趣的东西(地形算法)。我现在要停止浪费篇幅,只告诉你本章将要学习的内容:

- 如何将大型单图案纹理贴图应用到地形网
- 如何使用程序生成复杂的纹理贴图
各种地形“瓦片”
- 如何为地形添加细节纹理以添加更多内容
先前生成的纹理的细节

简单的纹理映射

我们将从一些简单的纹理映射开始。你会学习如何在整个地形网格上“拉伸”一个纹理。最多

有时,这种技术看起来很糟糕,当然,除非你有一个制作精良的纹理贴图,这就是我们将在下一节中进行的工作。现在重要的是你学会了如何拉伸纹理而不考虑最终结果会是什么样子。

为了在景观中拉伸单个纹理,我们将使景观中的每个顶点都落在 0.0f-1.0f 的范围内(纹理坐标的标准范围)。这样做比听起来更容易。首先,请看图 3.1。

如图 3.1 所示,地形网格的左下角(例如,我们将选择 256 × 256 的高度图分辨率),(0,0) 的纹理坐标为 (0.0f, 0.0f),地形的左上角(255, 255) 的纹理坐标为 (1.0f, 1.0f)。基本上,我们需要做的就是找出我们当前正在渲染的顶点并将其除以高度图分辨率。(正在做

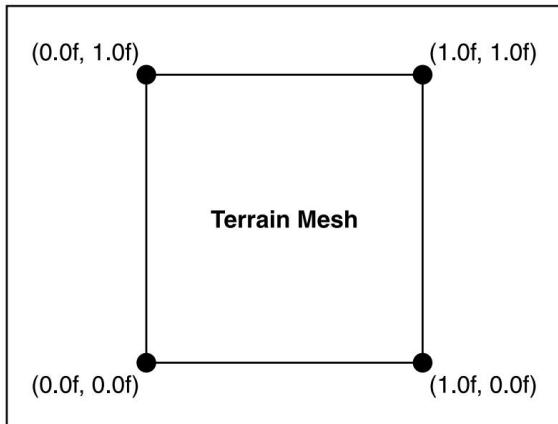


图 3.1 地形网格上的纹理坐标。

所以产生我们想要的范围内的值,0.0f-1.0f,而不需要我们跨过我们的边界。需要注意这一点很重要,因为我们确实希望在后面的部分中跳出前面提到的范围。)

在渲染每个顶点之前,我们需要计算三件事 x 、 z 和 $z+1$ 的纹理值,我将分别称为 $fTexLeft$ 、 $fTexBottom$ 和 $fTexTop$ 。以下是我们计算值的方法:

```
fTexLeft = ( 浮动 )x/m_iSize;
fTexBottom = ( 浮动 )z/m_iSize; fTexTop =
( 浮动 )( z+1 )/m_iSize;
```

并认为你认为这将是困难的!无论如何,我们需要对我们渲染的每个顶点进行前面的计算,然后将纹理坐标发送到我们的渲染 API。当我们渲染顶点(x, z)时,我们发送($fTexLeft, fTexBottom$)作为我们的纹理坐标,当我们渲染($x, z+1$)时,我们发送($fTexLeft, fTexTop$)作为我们的纹理坐标。查看 CD 中 Code\Chapter 3\demo3_1 中的图 3.2 和 demo3_1,看看您的劳动成果。

截图比我们在第 2 章 “Terrain 101”中的风景有更多的细节 (请注意,我删除了阴影),但很难辨别风景的实际形式。拉伸一个简单的纹理 (见图 3.3),即使在 demo3_1 中使用的纹理具有相当高的分辨率,也无法捕捉到我们希望在纹理贴图中拥有的细节量。

42 3. 纹理地形

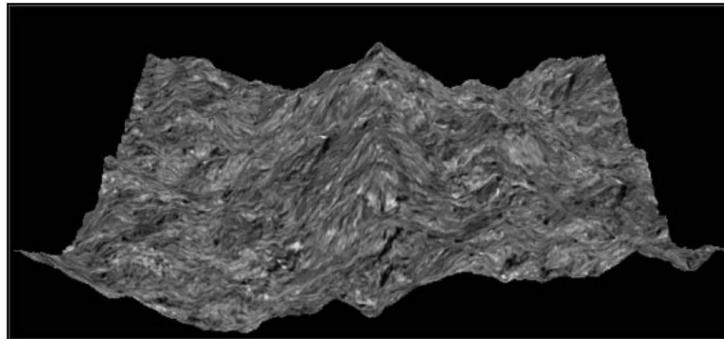


图 3.2 来自 demo3_1 的屏幕截图。

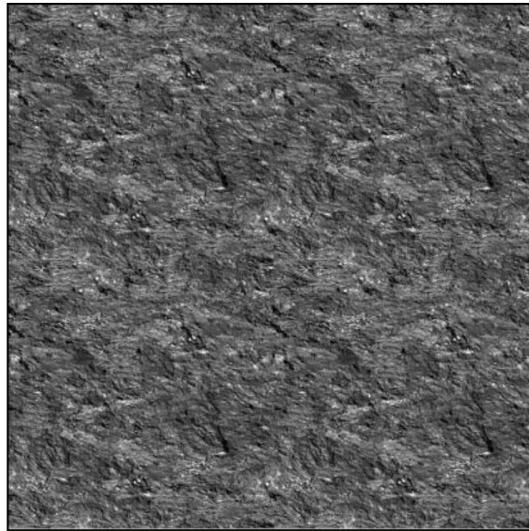


图 3.3 demo3_1 中使用的草纹理。

我们需要更多细节。我们想要一个类似于图 3.4 中的纹理贴图,它是使用一系列纹理“瓦片”(在本例中为泥土、草、岩石和雪)按程序生成的。

看到图 3.4 的纹理中显示了多少细节?该图的纹理有助于区分高山地区和低平原地区,这比图 3.3 中显示的单一草纹理要好得多。你需要知道如何生成一个非常酷的纹理,就像这里显示的那样。继续阅读!

程序纹理生成43

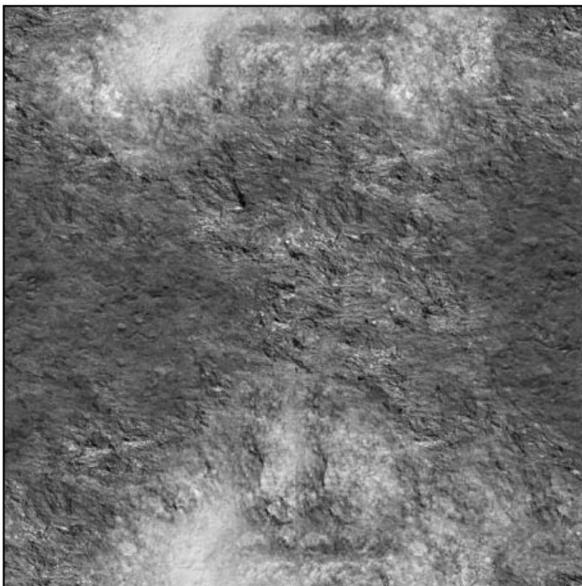


图 3.4 我们要用于演示的纹理类型。

程序纹理 一代

程序纹理生成是一种很酷且有用的技术,它是任何地形引擎的重要补充。在我们完成我们的程序纹理生成器之后,我们将让用户加载一系列他选择的两到四个图块。然后我们将调用我们的纹理生成函数。

(用户只需要知道他想要创建的纹理的大小。)就是这样!我们如何去创建我们的纹理生成函数?首先,您需要知道我们的实际目标是什么。我们将使用地形的高度图来生成与其一致的纹理。我们将遍历纹理贴图的每个像素,找到对应于该像素的高度,并确定每个纹理图块在该像素处的存在。(每个图块都有一个定义其影响区域的“区域”结构。)很少有图块在一个像素处100%可见,因此我们需要将该图块与其他图块“组合”(插入RGB颜色值)。结果将类似于图 3.5 中的样子,您可以在其中看到草和岩石块之间的插值效果。

44 3. 纹理地形

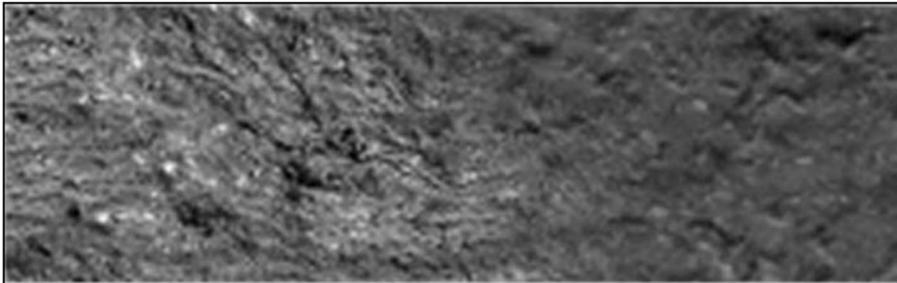


图 3.5 图 3.4 中纹理贴图的片段,显示了岩石和草块之间的插值。

区域系统要开始编码前面提到的过程,我们需要首先创建一个结构来保存每个图块的区域信息。

在这里应用的区域是一系列三个值,用于定义图块在我们的高度值范围 (0–255) 内的存在。这是我创建的结构:

```
结构STRN_TEXTURE_REGIONS
{
    诠释 m_iLowHeight; //最低可能高度 (0%) int m_iOptimalHeight; //最佳高
    度 (100%) int m_iHighHeight; //最高可能高度 (0%)
};

查看图 3.6 可以最好地解释每个值的作用。
```

为了便于说明,我们将`m_iLowHeight`等价于 63, `m_iOptimalHeight`等价于 128。计算`m_iHighHeight`的值需要一些简单的数学运算。我们想从`m_iOptimalHeight`中减去`m_iLowHeight`。然后我们想将`m_iOptimalHeight`添加到之前操作的结果中。我们已经设置了边界 (低:63,最佳:128,高:193),所以在图 3.6 中替换这些边界值。现在想象一下,我们正试图计算当前图块在高度为 150 处的存在程度。想象一下,考虑到我们创建的边界,该值将在图 3.6 中的位置。为了省去你想弄清楚的麻烦,请查看图 3.7。

程序纹理生成45

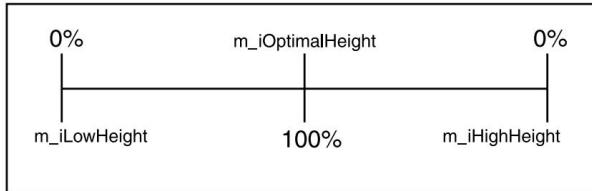


图 3.6 “纹理存在”线。

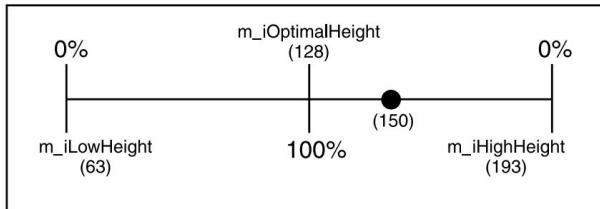


图 3.7 应用“texture-presence”线。

正如您从图像中看到的那样,纹理在该高度的存在
(150) 约为 70%。既然知道了这么多信息,
我们用它做什么?好吧,我们从纹理图像中提取 RGB 三元组并将其乘以 0.7f。
结果是我们在当前像素上想要多少纹理。

我们需要创建一个函数来计算区域百分比
为我们。这个功能比较简单。它需要两个简单的测试才能看到
给定的高度是否实际上在该区域的边界内;
如果不是,则退出该函数。接下来,我们需要弄清楚在哪里
高度位于该区域。是否低于最优值,高于最优值,
或等于最优值?最简单的情况是,如果高度是
相当于最优值;如果是,则当前图块在当前像素处具有 100% 的纹理存在,我们
不需要
完全不用担心插值。

如果高度低于最佳值,我们需要减少给定的
值到一个简单的分数。为此,我们采用给定的高度和
减去该区域的低边界。然后我们取最优边界值并减去低边界。然后我们

将第一次计算的结果除以第二次计算的结果,然后 BAM! 我们有我们的存
在百分比!

46 3. 纹理地形

这是刚刚讨论的代码：

```
//高度低于最佳高度
if( ucHeight<m_tiles.m_regions[tileType].m_iOptimalHeight ) {

    //计算给定瓦片区域的纹理百分比 fTemp1= ( float )m_tiles.m_regions[tileType].m_iLowHeight

        uc身高;
    fTemp2= ( 浮动 )m_tiles.m_regions[tileType].m_iOptimalHeight
        m_tiles.m_regions[tileType].m_iLowHeight;

    返回 (fTemp1/fTemp2) ;
}
```

最后一种情况是给定高度是否高于最佳边界。

这种情况的计算比高度低于边界时要复杂一些,但仍然不是很困难。这个解释在它的代码形式中比在文本中更容易看到,所以这里是代码：

```
//高度高于最佳高度 else
if( ucHeight>m_tiles.m_regions[tileType].m_iOptimalHeight ) {

    //计算给定瓦片区域的纹理百分比 fTemp1= ( float )m_tiles.m_regions[tileType].m_iHighHeight

        m_tiles.m_regions[tileType].m_iOptimalHeight;

    返回 (( fTemp1-( ucHeight
        m_tiles.m_regions[tileType].m_iOptimalHeight ) )/fTemp1 );
}
```

理论上,计算与低于最佳高度情况下的计算基本相同,只是我们必须能够将值降低到一个有意义的分数,因为 100% 低于高度,而不是高于高度。

而已!

平铺系统好的,现在您知道如何获得

一个纹理平铺和一个纹理像素的纹理存在。现在你需要应用你刚刚学到的所有东西来考虑所有四个纹理图块并创建一个完整的

程序纹理生成47

纹理贴图。不过,这比听起来容易得多,所以不要不知所措!

首先,我们需要创建一个可以管理所有纹理图块的纹理图块结构。我们不需要每个图块的太多信息;我们所需要的只是一个将纹理加载到其中的位置以及每个图块的区域结构。我们还希望跟踪加载的图块总数。考虑到所有这些要求,我创建了STRN_TEXTURE_TILES结构,如下所示:

结构STRN_TEXTURE_TILES

```
{
    STRN_TEXTURE_REGIONS m_regions[TRN_NUM_TILES];//纹理区域
    CIMAGE textureTiles[TRN_NUM_TILES]; //纹理瓷砖
    int iNumTiles;
};
```

接下来,您需要一些纹理平铺管理功能。我有加载和卸载单个瓦片的函数,以及一次卸载所有瓦片的函数。这些函数实现起来很简单,所以我不会在这里展示它们的片段。如果您有兴趣,只需查看代码。除此之外,您已经准备好编写纹理生成函数了!

要启动生成功能,我们需要弄清楚实际加载了多少瓦片。(我们希望用户能够在没有加载所有四个图块的情况下生成纹理。)完成之后,我们需要重新循环通过图块来确定每个图块的区域边界。

(我们希望平铺区域在 0-255 范围内均匀分布)。这是我这样做的方法:

```
iLastHeight= -1;
for(i=0;i<TRN_NUM_TILES;i++)
{
    //我们只想在我们//实际上加载了一个图块时执行这些计算
    if( m_tiles.textureTiles[i].IsLoaded() )

    {
        //计算三个高度边界 m_tiles.m_regions[i].m_iLowHeight=
        iLastHeight+1; iLastHeight+= 255/m_tiles.iNumTiles;
```

48 3. 纹理地形

```
m_tiles.m_regions[i].m_iOptimalHeight= iLastHeight;

m_tiles.m_regions[i].m_iHighHeight= (iLastHeight
    m_tiles.m_regions[i].m_iLowHeight)+iLastHeight;
}

}
```

这里唯一看起来有点奇怪的是我们计算m_iHighHeight 的最后一段,即使它看起来也不那么奇怪,因为我们之前已经解释过了。(如果它看起来很奇怪,请回到本节开头,我解释了区域边界。)

创建纹理数据现在是时候创建实际的纹理数据了。为此,我们需要创建三个不同的for循环:一个用于纹理贴图的Z轴,一个用于X轴,一个用于穿过每个图块。(这将是此函数中的第三个图块循环。)我们还需要创建三个变量,以保持我们正在计算的每个像素的当前红色、绿色和蓝色分量的总和。这是实际纹理生成开始时的样子:

```
for(z=0;z<uiSize;z++)
{
    对于 (x=0;x<uiSize;x++)
    {
        //将我们的总颜色计数器设置为 0.0f
        fTotalRed = 0.0f;
        fTotalGreen= 0.0f;
        fTotalBlue = 0.0f;

        //循环遍历图块 //第三次在这个函数中

        for(i=0;i<TRN_NUM_TILES;i++)
        {
            //如果瓦片被加载,我们可以执行计算 if( m_tiles.textureTiles[i].IsLoaded() )

        }
    }
}
```

接下来,我们需要从纹理(在当前像素处)提取RGB值到我们的临时RGB unsigned char变量中。完成后,我们需要确定当前图块在

程序纹理生成49

当前像素（使用我们之前创建的函数），将临时 RGB 变量乘以结果，并将其添加到我们的总 RGB 计数器中。现在我们需要将前面的解释放入代码中：

```
//在我们在GetTexCoords中得到的坐标处获取纹理中的当前颜色 m_tiles.textureTiles[i].GetColor( uiTexX,
uiTexZ,
```

```
    &ucRed, &ucGreen, &ucBlue );
```

```
//获取该图块的当前坐标的混合百分比 fBlend[i]=RegionPercent( i, InterpolateHeight( x, z, fMapRatio ) );
```

```
//计算将使用的RGB值
```

```
fTotalRed += ucRed*fBlend[i];
fTotalGreen+= ucGreen*fBlend[i];
fTotalBlue += ucBlue*fBlend[i];
```

在我们遍历所有四个图块之后，我们为正在创建的纹理中的像素设置颜色，然后为下一个像素重做整个事情。当我们完全完成为纹理生成颜色值时，我们创建纹理以与我们的图形 API 一起使用，然后我们就设置好了！

改进纹理生成器好吧，我撒谎了。我们还没有准备好。我们的纹理生成函数在其当前形式中存在一些问题。这些问题如下：

- 我们只能创建分辨率等于或低于我们的高度图。
- 如果我们解决了这个问题，那么我们只能创建一个分辨率等于或低于我们最小纹理图块的分辨率的纹理。

然而，这两个问题都相对容易解决。让我们从高度图分辨率问题开始。

摆脱高度图分辨率依赖我们需要让用户选择他想要的任何纹理大小。（嗯，几乎任何尺寸。我们希望尺寸是 2 的幂。）

在我们的纹理生成功能的早期，在我们进入庞大的系列之前

50 3. 纹理地形

对于循环,我们需要计算出高度图与纹理图像素的比率,可以这样完成:

```
fMapRatio= ( 浮动 )m_iSize/uiSize;
```

然后我们需要创建一个函数来插入我们从高度图中提取的值。我们将把这个插值函数分成两部分:一部分用于 X 轴,另一部分用于 Z 轴。然后我们将获得两个部分的结果的平均值,这是我们插值的高度值。这可能不是处理事情的最佳方式,但它很有效,而且运行速度很快!

对于这个函数,我们需要三个参数。前两个参数是我们获取信息的未缩放 (x, z) 坐标。

这将相当高,并且很可能超出高度图的范围。第三个参数是我们计算高度与纹理贴图像素比(fMapRatio) 的变量。在函数内部,我们将通过比率变量缩放给定的 (x, z) 坐标,并将这些坐标用于大多数函数。要计算沿 X 轴的插值,我们将这样做:

```
//设置中间边界 ucLow=
GetTrueHeightAtPoint( ( int )fScaledX, ( int )fScaledZ );

//设置高边界
if( ( fScaledX+1 )>m_iSize )
    返回 ucLow;
别的
    ucHighX= GetTrueHeightAtPoint( ( int )fScaledX+1,
                                    (int)fScaledZ);

//计算插值 (对于X轴) fInterpolation= ( fScaledX-( int )fScaledX );
UCX
    = ( ( ucHighX-ucLow )*fInterpolation )+ucLow;
```

如您所见,我们要做的第一件事是获得低高度。然后我们检查高度图上的下一个高度是否在高度图上。如果不是,那么我们必须满足于低价值。

如果下一个高度在高度图上,那么我们可以得到它并准备插入两个值。我们得到浮点缩放的 x 值和unsigned char 缩放的 x 值之间的差异。

(它将具有较低的准确度,这将定义我们将使用的插值量。)在接下来的计算中,我们计算沿 X 轴的插值。然后我们对 Z 轴做同样的事情,将两个计算的结果相加,然后除以 2。

这里的所有都是它的!

摆脱瓷砖分辨率依赖好的,我们快到了。我们还有一件事要弄清楚,那就是我们如何消除设置在用户身上的图块大小边界。解决方案是如此明显,以至于您可能会想,“我为什么没有想到这一点?”好吧,相信我,我花了很长时间才找到解决方案,所以不要难过。我们需要做的就是重复瓷砖!我创建了一个简单的函数,它将为我们提供“新”纹理坐标,为我们重复我们的纹理。这是功能:

```
void CTERRAIN::GetTexCoords( CIMAGE 纹理,
                           无符号整数* x, 无符号整数* y)
{
    无符号 int uiWidth = texture.GetWidth(); unsigned int uiHeight=
    texture.GetHeight(); 诠释 iRepeatX = -1; int iRepeatY = -1; 诠释我 = 0;

    //循环直到我们计算出瓷砖//重复了多少次 (在X轴上)while(iRepeatX== -1)

    {
        我++;

        //如果x小于总宽度,
        //然后我们找到了一个赢家!
        if( *x<( uiWidth*i ))
            iRepeatX= i-1;
    }

    //准备计算Y轴上的重复
    我= 0;
```

52.3. 纹理地形

```
//循环直到我们计算出瓷砖重复了多少次// (在Y轴上)

而 (iRepeatY== -1){

    我++;

    //如果y小于总高度,那么我们就有宾果游戏! if( *y<( uiHeight*i ) ) iRepeatY= i-1;

}

//更新给定的纹理坐标 *x= *x-( uiWidth*iRepeatX );
*y= *y-( uiHeight*iRepeatY );

}
```

这个函数的大部分由两个while循环组成,其主要目标只是试图找出纹理在到达作为参数给出的坐标之前重复了多少次。弄清楚之后,我们只是缩小给定的坐标,使它们回到纹理的值范围内。(我们不想尝试提取完全超出纹理范围的信息。

那会导致错误,错误是不好的。)

而已!我们的纹理生成功能现已完成!查看图 3.8。在演示中,您会注意到菜单中有一个名为 Texture Map 的新字段。在该字段中,您可以生成更高分辨率的新纹理或将当前纹理保存到演示目录。

说到演示,您可以在 CD 的 Code\Chapter 3\demo3_2 中看到您在 demo3_2 中的所有辛勤工作。只需在 Microsoft Visual C++ 中打开演示的工作区并开始享受乐趣!

使用细节贴图

1024×1024 是一个相当大的数据量,用于实现我们希望在纹理中拥有的细节量。必须有另一种方法来实现我们想要的细节而不浪费资源。好吧,不要再想了!为您的景观添加更多细节的一种很酷的方法是使用细节图。细节贴图是一种类似于图 3.9 中的灰度纹理,它在景观上重复多次,并添加了一些很酷的细微差别,例如裂缝、颠簸、岩石和其他有趣的东西。

使用细节贴图53

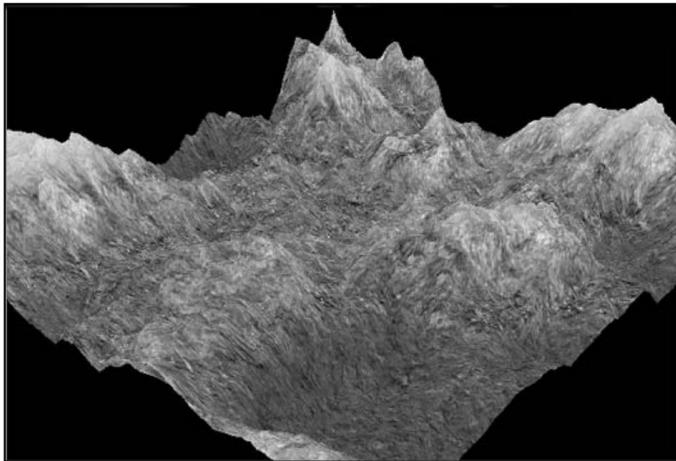


图 3.8 demo3_2 截图中使用的纹理,分辨率为 1024×1024 。



图 3.9 详细地图示例。

向地形引擎添加细节地图支持是一个简单的过程。

添加一些用于加载/卸载细节图的管理功能,该功能允许用户决定他希望地图在景观中重复多少次,然后您所要做的就是稍微编辑您的渲染代码。最艰难的决定是决定

54.3. 纹理地形

无论是硬件多纹理还是只制作两个单独的渲染通道。因为地形网格可能变得相当大,所以最好的选择是坚持使用硬件多纹理。实现硬件多纹理超出了本书的范围,但如果你不知道如何使用图形 API 来实现它,那么实现它是一件简单的事情,你应该学习它。如果您不知道学习 API 的好地方,请查看由 Premier Press 出版的OpenGL 游戏编程(Astle/Hawkins) 或Special Effects Game Programming with DirectX 8.0 (McCuskey) 如果我这样做的话,这是一个很棒的出版商说我自己!

要编辑渲染代码,只需将基础颜色纹理 (例如我们之前生成的纹理)设置为第一个纹理单元,然后将细节纹理设置为第二个纹理单元。记住如何

颜色纹理的纹理坐标是这样计算的?

```
fTexLeft = ( 浮动 )x/m_iSize;
fTexBottom= ( 浮动 )z/m_jSize;
fTexTop = ( 浮动 )( z+1 )/m_iSize;
```

好吧,我们只需对这些计算稍作修改即可获得我们的细节纹理的纹理坐标:

```
fTexLeft = ( float )( x/m_iSize )*m_iRepeatDetailMap; fTexBottom= ( float )( z/
m_iSize )*m_iRepeatDetailMap ); fTexTop = ( float )( ( z+1 )/
m_iSize )*m_iRepeatDetailMap );
```

`m_iRepeatDetailMap`是用户希望细节图在景观中重复的次数。细节映射最困难的部分是使用图形 API 设置多纹理,但如果您使用的是 OpenGL,我会为您设置好一切。(是的,我知道我是个好人,如果你觉得有必要报答我,我的生日是 3 月 11 日!)要查看详细地图对地形的影响,请查看地形与地形的区别使用没有细节贴图的 256 × 256 程序纹理 (图 3.10 的左侧)和使用相同的 256 × 256 纹理的地形,除了细节贴图 (图 3.10 的右侧)。

看看右边的图像有多少细节?最好的部分是添加大量细节很简单。查看 CD 上的 `Code\Chapter 3\demo3_3` 中的 `demo3_3`,它显示了新的详细地图代码的运行情况。其余演示中唯一更改的控件是 T 关闭细节映射,D 重新打开细节映射。(默认情况下,细节映射是打开的。)

参考文献55

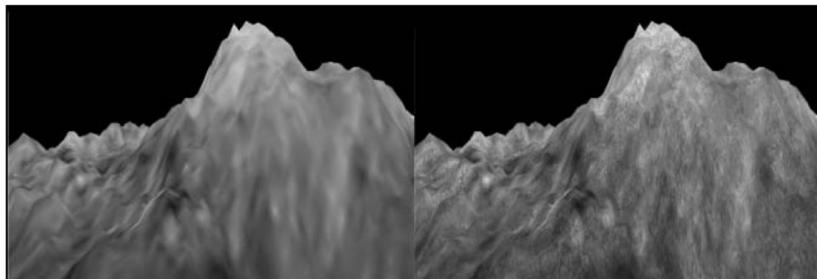


图 3.10 两张地形图对比,一张没有细节图(左),一张有细节图(右)。

总结我们在本章中学到了很多

关于纹理地形的知识。我们从简单的纹理开始(在整个景观中拉伸单个纹理),然后我们通过程序纹理生成将事情推向了高潮。我们以一种简单但很酷的技术结束,称为细节映射。在下一章中,我们将学习使地形更逼真的下一步:照明。如果您碰巧喜欢纹理技术,您可能想查看 Tobias Franke 的文章“Terrain Texture Generation”¹或 Yordan Gyurchev 的文章“Generating Terrain Textures”²。您也可能对 Jeff Lander 的文章“Terrain Textures”³感兴趣,提出了一种动态纹理平铺解决方案。

参考

1 弗兰卡,托拜厄斯。“地形纹理生成。” 2001. http://www.flipcode.com/tutorials/tut_proctext.shtml。

2 久尔切夫,约尔丹。“生成地形纹理。” 2001. http://www.flipcode.com/tutorials/tut_terrtext.shtml。

3 兰德,杰夫。“地形纹理。” Delphi3D-快速 OpenGL 开发。 2002. <http://www.delphi3d.net/articles/viewarticle.php?article=terraintex.htm>。

第4章

灯光 地形

58 4. 照明地形

纹理地形为我们带来了新的细节水平。
地形和光照都带来了全新的真实感。

问题是这样的：我们如何尽可能快地照亮我们的地形

同时仍然保持高水平的现实主义？好吧，所有的技术

我将教你们所有快速（如果粗糙）照明地形的方法。

我不会涉及复杂的全局照明算法

（尽管我会向您指出一些可以获取其中一些信息的地方）因为逼真的地形照明可能

单独覆盖整本书。话虽如此，这是议程

本章：

- 基于高度的照明
- 五金照明
- 将光照射贴图应用到地形
- 超酷的斜坡照明算法

我会尽量简短地讨论照明，因为我知道

你（或者如果你不这样做，那么我知道我愿意）想要开始

我将在接下来的三章中介绍很酷的地形算法。我们走吧！

基于高度的照明

基于高度的照明简单且不切实际，但它是一种照明，所以我想我至少会简要介绍一下。我们在第 2 章“Terrain 101”的所有演示中都使用了基于高度的光照，所以你已经使用了

之前，即使你不知道。

基于高度的照明就是这样 基于高度的照明

顶点。高顶点（基于地形补丁的高度数据）

高度数据）比低顶点更亮，仅此而已。

我们需要做的就是使用我们的GetTrueHeightAtPoint函数（CTERRAIN类的成员）来提取像素点的亮度

当前(x, z)位置（值将在 0-255 范围内）从

高度图，这就是我们的亮度值。就这么简单！

图 4.1 强化了这一概念。

基于高度的照明59

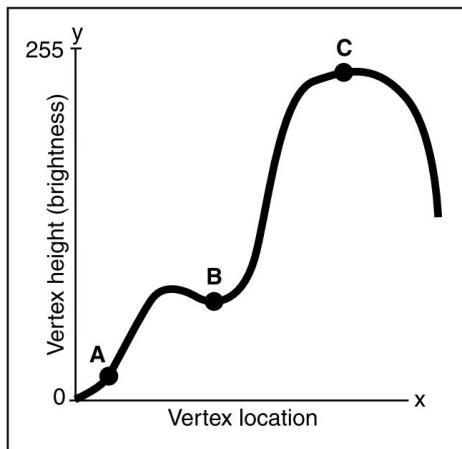


图 4.1 基于高度的照明的视觉解释。

在图中,顶点 A 几乎是黑色的,顶点 B 会更亮一些,而顶点 C 会被完全照亮 (白色)。

好了 三段和一张图对基于高度的照明的完整解释!

现在您知道什么是基于高度的光照,但是如何计算代码中的光照值呢?好吧,这很简单,考虑到你已经加载了一个高度图。例如,假设您正在尝试计算地形中顶点 (157, 227) 的亮度。好吧,顶点的亮度只是您从高度图中提取的高度值。

```
ucShade=GetTrueHeightAtPoint(157, 227);
```

ucShade是我们存储光照值的变量, GetTrueHeightAtPoint从高度图中提取信息,在本例中位于顶点 (157, 227),范围为 0 (暗)到 255 (亮)。

现在,让我们为灯光添加一些颜色!

为光源着色我们并不总是希望我们的照明颜色为灰度 (从黑到白)。大多数时候,我们希望我们的照明能够针对各种情况进行着色。例如,如果这是一个万里无云的夜晚,用户将会体验到美丽的日落,所以我们希望我们的灯光颜色是橙色、粉红色或紫色的阴影。我们需要创建一个向量

60 4. 照明地形

对于我们的灯光颜色信息和一个简单的函数来设置灯光的颜色。（我们希望光照值在 0.0f-1.0f 的范围内。你会在一秒钟内找到原因。）完成之后，我们可以取之前检索到的光照值并将其乘以每个 RGB 光颜色向量中的值使用此等式：

强度=阴影*颜色

现在，使用该等式，我们可以应用它来计算 RGB 颜色分量，然后将它们发送到渲染 API：

```
ColorToAPI( ( unsigned char )( ucShade*m_vecLightColor[0]),
            ( unsigned char )( ucShade*m_vecLightColor[1]), ( unsigned char )
            ( ucShade*m_vecLightColor[2] ));
```

`ucShade`是我们之前计算的亮度值，`vecLightColor`是我们灯光的颜色。

现在查看 `demo4_1`（在 CD 上的 `Code\Chapter 4\demo4_1` 下）和图 4.2。如果您需要重新了解演示的控件，请查看表 4.1。

当您查看该图时，您会注意到地形的较低区域相当暗，而地形的高区域则很亮。

这正是基于高度的照明所做的：高区域是明亮的，而低区域是黑暗的。这个算法有什么问题？首先，这是非常不现实的。该算法不考虑

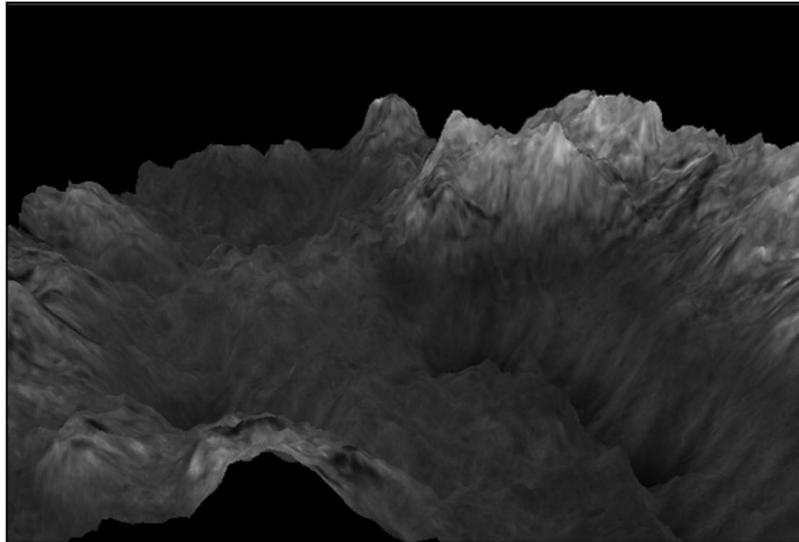


图 4.2 来自 `demo4_1` 的屏幕截图。



表 4.1 演示 4_1 的控件

钥匙	功能
逃生/Q	退出程序
向上箭头	前进
向下箭头	往后退
右箭头	向右扫射
左箭头	左扫射
吨	切换纹理映射
D	切换细节映射
在	以线框模式渲染
+	以实体/填充模式渲染
+ / -	增加/减少鼠标灵敏度
] / [增加/减少移动灵敏度

太阳有可能直接照射在“倾角”地形（低高度区域），这将使该区域非常明亮。这个问题如图 4.3 所示。

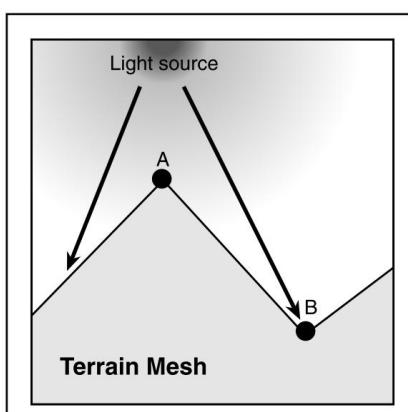


图 4.3 基于高度的照明的一个问题。

62 4. 照明地形

您会看到,在图 4.3 中,太阳同时到达顶点 A 和 B,但是根据我们使用基于高度的照明技术照亮地形的方式,顶点 A 会是亮的,而顶点 B 会是暗的,这是不正确的(如图所示)。

这种技术的第二个问题是,它为您提供的地形照明外观方式几乎没有自由度。现在我们需要继续讨论更通用和更现实的地形照明方式。

硬件照明这种技术有两个主要问题。首先,它高度依赖 API,所以我不会向您展示代码或给您演示。

其次,它对于动态地形网格毫无用处 我们将在接下来的三章中使用这种网格。由于这些问题,我在这里只给你一些基本的实现细节。

硬件照明要求您计算渲染的每个三角形的表面法线。执行此操作的最佳时间是在演示的预处理部分;这样,计算就不会使程序陷入困境。计算出法线后,只需将其发送到要渲染的当前三角形的 API,就完成了。

警告

在执行任何操作之前,请确保您使用 API 正确设置了硬件照明。有时,硬件照明设置起来确实很麻烦,所以如果您的地形在第一次尝试实现它时没有被照亮或被错误地照亮,请不要太惊讶。您需要确保您有自定义光源(具有正确的衰减、漫反射/镜面反射/环境值等)。设置好灯光后,请确保您启用了 API 的光源和照明组件。很多人来找我询问有关硬件照明的问题,其中 75% 的人忘记启用他们的光源!不要成为一个统计数字。

光照贴图63

这种技术非常适用于静态地形网格,例如我们在过去两章中使用的那种,以及我们在本章中使用的那种。它使动态照明和日/夜模拟变得轻而易举。然而,由于硬件照明主要是基于顶点的,动态地形网格在硬件照明下看起来并不好。(动态网格具有不断变化的顶点。)这就是我们对硬件照明的讨论。希望你没有眨眼。

光照贴图

我们将在本书中不断使用光照贴图。光照贴图与高度贴图(在第2章中讨论)完全相同,只是光照贴图不包含高度信息,而是仅包含光照信息。光照贴图的加载、保存、卸载的所有代码都与高度贴图的相应程序相同(除了光照贴图操作函数处理的变量与高度贴图函数不同),所以我不会浪费你的宝贵再次遍历每个函数。我们的光照贴图信息将存储在灰度RAW纹理中,就像我们的高度贴图一样,除了光照贴图中的信息仅与光照有关。例如,查看图4.4中的高度图和光照图,然后查看图4.5中实现的结果。看看光照贴图如何影响地形光照?

看看图4.5中的灯光是如何与图4.4中的光照贴图完全一样的球形?这就是我们使用光照贴图的原因:为一片地形定义我们想要的准确光照类型。而且因为您可以预先创建光照贴图,所以您可以使用各种算法来生成它们。

您可以通过多种方式生成光照贴图。其中一些方法是

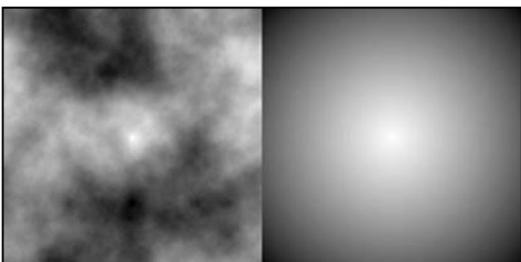


图4.4图4.5中地形的高度图(左)和光照图(右)。

64 4. 照明地形

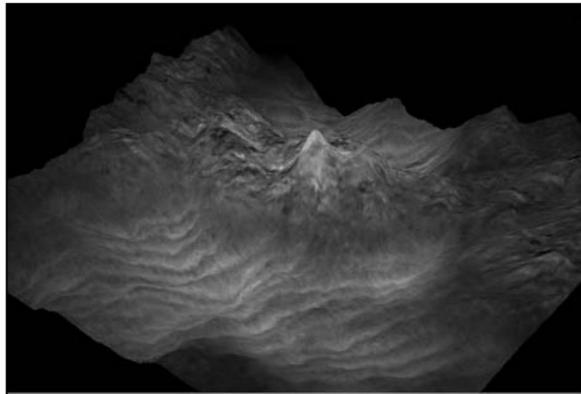


图 4.5 从图 4.4 中的高度图和光照图创建的地形。

复杂但美观的全局照明技术。在下一节中,我将向您展示一种创建光照贴图的方法。

在以与加载高度图相同的方式加载光照贴图后,您需要创建一个函数来提取给定像素的亮度:

```
内联 unsigned char GetBrightnessAtPoint( int x, int z ) { return
( m_lightmap.m_ucpData[( z*m_lightmap.m_iSize )+x] ); }
```

还记得我们在 demo4_2 中如何使用GetTrueHeightAtPoint获取基于高度的光照的亮度信息吗?我们所要做的就是用对GetBrightnessAtPoint的调用替换该调用,然后我们就准备好了!看看所有这些照明技术有多简单?查看 CD 上 Code\Chapter 4\demo4_2 下的 demo4_2,并尝试创建一些您自己的小高度图,看看结果如何。我在图 4.6 中创建了一个有趣的小光照贴图,结果如图 4.7 所示。

好像以前并不明显,我正遭受着一种极端的结合,即我手上的时间太多,而这本书太有趣了!

光照贴图在游戏中非常重要,以至于我有种扩展它的一些更高级功能的冲动。我介绍了将光照贴图“粘贴”到地形上所需的所有内容,但更高级的光照贴图概念集中在光照贴图的生成上。(稍后我将向您展示一种这样的算法。)

光照贴图65



图 4.6 用于创建图 4.7 中地形的
光照贴图。



图 4.7 从图 4.6 中的光照贴图创建的地形。

许多游戏,例如Max-Payne和Quake 2,使用一种称为辐射度的方法。(如果您想了解该技术,请访问 <http://freespace.virgin.net/hugo.elias/radiosity/radiosity.htm。>)

请注意,我提到的游戏都是基于室内的游戏,而地形是一个户外主题,正如您可能猜到的那样,这意味着我们必须找到另一种技术来计算我们的光照贴图。对我们来说幸运的是,有许多不同的技术可供使用(实际上几乎是令人困惑的数量)。我将提供一个这样简单的算法,但要知道它是一个简单的算法,并且不如现有的其他一些全局照明技术强大,我将在本文末尾提及其中之一章节。

66 4. 照明地形

斜坡照明

现在让我把它从我的系统中拿出来 :这是我很长时间以来遇到的最酷的算法之一。它使用起来非常简单 , 并且提供了清晰的效果。斜坡照明 1 是一种简单的照明技术 , 可根据顶点相对于附近顶点的高度对顶点进行着色。

好的 , 斜坡照明很酷 , 但它是如何执行的 ?

为了倾斜光照地形 , 我们将从当前顶点旁边的顶点检索高度 (方向将由光照方向决定) , 然后减去当前顶点的高度。

基本上 , 我们正在检查其他顶点是否会在当前顶点上投射阴影。我认为这是一个现实生活中的例子的最佳时机。假设您站在一栋从您的角度遮挡阳光的大型建筑物前。如图 4.8 所示 , 建筑物会在你身上投下阴影。

正如您在图中看到的 , 由于大型建筑物挡住了光源 , 光源的光线不会到达您的位置。最终结果是你将站在一个阴暗的区域 , 让接受光线的人看起来更暗。这与我们试图在斜坡照明中实现的概念相同 着色顶点

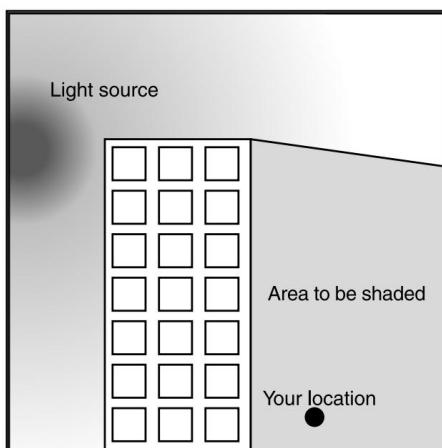


图 4.8 建筑类比。

斜坡照明67

光源的光线被前面的较高顶点阻挡。图 4.9 进一步解释了这个概念。

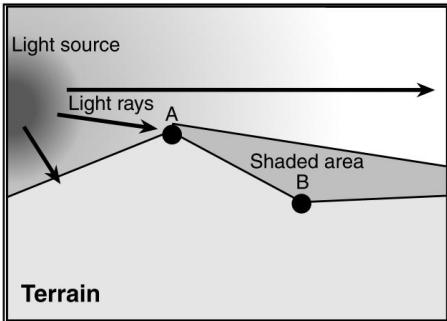


图 4.9 斜坡照明的视觉示例。

如图所示,光源发出各种光线(实际上几乎是无限量),但在这种情况下,它们都不会到达顶点 B,因为顶点 A 阻挡了光线。仅仅因为顶点 B 没有接收到直射光线并不意味着它是完全黑暗的;其他顶点接收到的一些光“渗出”到顶点,稍微照亮它。顶点永远不会完全黑暗。

这个算法有一个小缺陷,那就是当你设置光的方向时,它必须以 45 度为增量。例如,图 4.10 的左侧被方向为 (1, 1) 的光照亮。如果我们想将灯光向左移动,我们必须将灯光更改为 (0, 1) 的方向,这将导致灯光向左移动 45 度,而不是像一次 2-5 度。

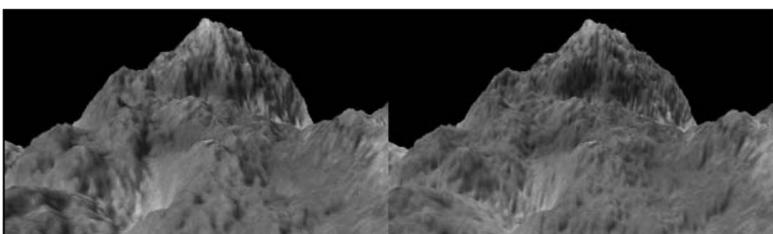


图 4.10 倾斜光照的地形,光照方向为 (1, 1) (左),光照方向为 (0, 1) (右)。

68 4. 照明地形

现在,如果您会注意到,这些图像之间实际上并没有太大的区别。因此,您可以轻松地实时更改灯光的方向。大多数用户不会注意到阴影中的轻微“跳跃”。

创建斜坡照明系统

在开始编写斜坡照明代码之前,我们需要向CTERRAIN类添加一些变量。(如果你认为这个类现在充满了很多额外的功能,那就等到以后吧!)我们需要能够为我们的地形定义最小/最大亮度,因为正如我之前所说,阴影顶点很少曾经完全变暗。我们还需要灯光的柔和度和灯光方向的变量,如果我们想要获得逼真的效果,这两个变量都需要使用。此外,我们还需要一个功能,让用户轻松自定义斜坡照明系统的参数:

```
内联无效 CustomizeSlopeLighting (int iDirX,int iDirZ,
                                    浮动 fMinBrightness,浮动
                                    fMaxBrightness,
                                    浮动 fSoftness )
{
    //设置灯光方向 m_iDirectionX= iDirX;
    m_iDirectionZ= iDirZ;

    //设置最小/最大着色值 m_fMinBrightness= fMinBrightness;
    m_fMaxBrightness= fMaxBrightness;

    //设置灯光的柔和度 m_fLightSoftness=
    fSoftness;
}
```

动态创建光照贴图在我们进一步讨论之前,您还记得我说过最关键的光照贴图算法包括光照贴图生成吗?好吧,现在我们将创建一个函数,该函数将生成一个光照贴图供我们的地形使用。您可以选择计算每一帧的光照(使用我介绍的算法,这个过程并没有那么慢)

斜坡照明69

这里),但最好在演示开始时计算一次,然后在需要时再次计算照明。

这是我为此创建的函数的前半部分:

```
无效 CTERRAIN::CalculateLighting( 无效 )
{
    浮动 fShade;
    整数 x, z;

    //已经提供了光照贴图,不需要创建一个 if( m_lightingType==LIGHTMAP )

    返回;

    //如果需要就分配内存 if( m_lightmap.m_iSize!
    =m_iSize || m_lightmap.m_ucpData==NULL ) {

        //删除旧数据的内存 delete[] m_lightmap.m_ucpData;

        //为新的光照贴图数据缓冲区分配内存 m_lightmap.m_ucpData= new
        unsigned char [m_iSize*m_iSize]; m_lightmap.m_iSize= m_iSize;

    }

    //遍历所有顶点
    for(z=0;z<m_iSize;z++)
    {
        对于 (x=0;x<m_iSize;x++)
        {
            //使用基于高度的光照,微不足道
            if( m_lightingType==HEIGHT_BASED )
                SetBrightnessAtPoint( x, z,
                    GetTrueHeightAtPoint(x, z));
    }
}
```

至此,你应该能够理解一切。

我们首先检查用户是否在使用预制光照贴图。如果他是,那么我们不想覆盖光照贴图中的信息。然后我们需要看看是否需要为光照贴图分配内存。之后,我们开始循环遍历所有顶点 光照贴图需要与高度贴图大小相同 并检查用户是否使用基于高度的光照。如果他

70 4. 光照地形

是,然后我们将当前像素设置为与高度图中相同的像素。该函数的其余部分与坡度照明有关,因此我将分两节进行描述:

```
//使用斜坡照明技术 else
if( m_lightingType==SLOPE_LIGHT ) {

    //确保我们不会通过这样做而跨过数组边界

    如果 (z>=m_iDirectionZ && x>=m_iDirectionX)
    {
        //使用“斜坡照明”计算着色值 //算法 fShade= 1.0f-( GetTrueHeightAtPoint( x-
        m_iDirectionX, z-m_iDirectionZ )

                GetTrueHeightAtPoint(x, z))/m_fLightSoftness;
    }
}
```

这是进行大部分斜坡照明计算的地方。

如您所见,我们用当前顶点减去当前顶点之前的顶点高度 (在用户指定的方向上)。

我们试图查看前一个顶点投射了多少阴影。

然后我们将该值除以光柔和度,并从除法后的值中减去 1.0f。我真的没有谈论过光线的柔和度如何影响事物,所以请查看图 4.11,我在其中使用三个不同的柔和度级别截取了屏幕截图。

现在,函数的其余部分:

```
//如果我们跨过边界,那么就返回一个非常亮的颜色值 (白色)否则
```

```
fShade=1.0f;
```

```
//将着色值限制在最小/最大 //亮度边界 if( fShade<m_fMinBrightness )
fShade= m_fMinBrightness;如果 ( fShade>m_fMaxBrightness )
fShade= m_fMaxBrightness;
```

```
//为我们的光照贴图设置新的亮度  
SetBrightnessAtPoint( x, z, ( unsigned  
                           char )( fShade*255 ) );  
}  
}  
}  
}
```

在本节中,我们将fShade钳制在最小/最大亮度边界,然后在光照贴图中设置当前点的亮度。在 demo4_3 (在 CD 上的 Code\Chapter 4\demo4_3 下) 中,我添加了一个很酷的新对话框 (参见图 4.12), 让您可以动态地完全自定义斜坡照明系统。玩得开心!

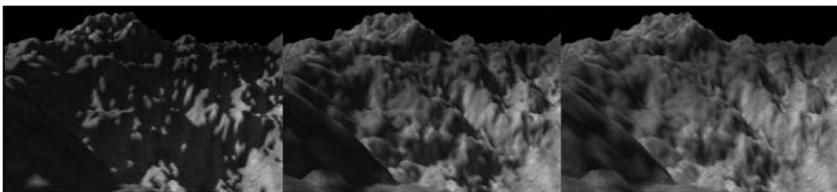


图 4.11 不同级别的光线柔和度:值 1 (左)、10 (中) 和 15 (右)。

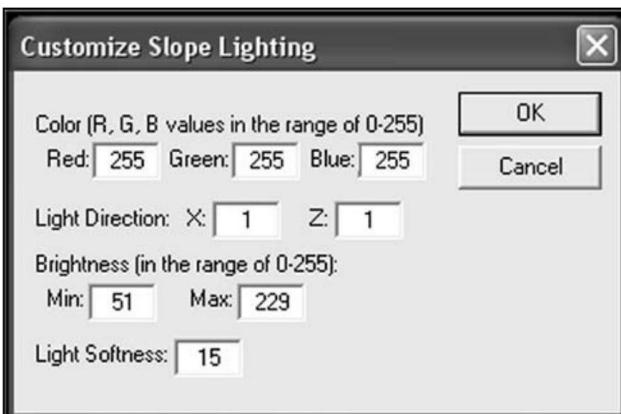


图 4.12 demo4_3 中的自定义斜坡照明对话框。

72 4. 照明地形

概括

本章快速介绍了一些简单的照明技术,这些技术可以为您的地形增添新的真实感。我们讨论了基于高度的光照、硬件光照、光照贴图和斜坡光照。斜坡照明可能是简单地形演示的最佳选择。我没有时间提及一种非常酷的全局照明技术,即 Hoffman 和 Mitchell 的文章 “Real-Time Photorealistic Terrain Lighting”,但如果您对地形照明感兴趣,绝对值得一看。无论如何,你最好做好准备 你即将进入硬核地形编程部分,其中包含有关高级地形算法的各种信息。

参考

- 1 范诺兰,查理。“斜坡照明地形。” 2002. <http://www.gamedev.net/reference/articles/article1436.asp>。

第二部分

先进的 地形 编程

5 CLOD 受损的 Geomipmapping

6 攀登四叉树

7 你可以在哪里漫游

8 总结:特别
效果及更多

附录 CD 上的内容

第 5 章

Geomip

映射

为了

CLOD

受损

76.5. CLOD 受损的 Geomipmapping

呜呼!您现在将了解有关硬核地形的所有信息

事实上,这是一个谎言。选择我将在本章中解释的三种算法是因为它们的简单性和效率。而且,在本书中,我打算让你免于冗长的介绍,而只是简单地告诉你本章的议程:

- 详细说明连续详细级别的含义 ■ geomipmapping 背后的理论 ■
- geomipmapping 的实现方法

为简单起见,我将议程分为三个部分。尽管有这样的细分,这一章还是相当大的。但是,不要让本章的篇幅吓到您;内容将一如既往地以有趣和简单的方式呈现。请注意,我正在稍微改变学习方式。与前几章相比,第 5、6 和 7 章更侧重于算法解释和伪代码。在后面的这些章节中,我仍然为您提供了自己的演示和实现,但实现很简单,您应该只结合文本使用它们。话虽如此,让我们开始吧。

CLOD 地形 101

在本书中,您已经多次听说过连续详细级别(CLOD)一词,但现在是时候告诉您它的实际含义了。一句话,CLOD 算法是一种动态多边形网格,它为需要更多细节的区域“提供”额外的三角形。这是一个简单的陈述,但在本节结束时你会了解更多关于 CLOD 的内容,并且在本章结束时你会了解更多。如果您还不了解 CLOD 是什么,请不要担心。

为什么要打扰 CLOD Terrain?

CLOD 算法需要更多的研究,更难编码,并且比普通的蛮力实现占用更多的 CPU 周期。

CLOD 地形 101 77

考虑到这一点,您为什么要使用 CLOD 算法呢?

这真的很简单:创建一个更逼真、更详细,最重要的是,更快的地形补丁。

添加了更多细节

需要更多细节的地方

CLOD 的基本思想之一是我们希望在需要的地方添加更多细节 (更多三角形)。例如,如果我们有一块相当平滑的地形 (见图 5.1),我们平均需要更少的三角形,而不是更混乱的地形 (见图 5.2)。

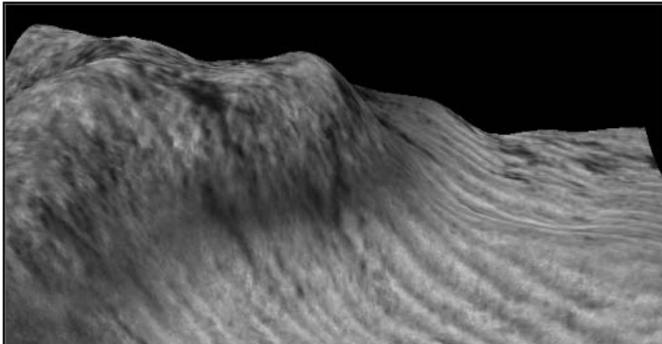


图 5.1 渲染这块平滑的地形需要几个三角形。

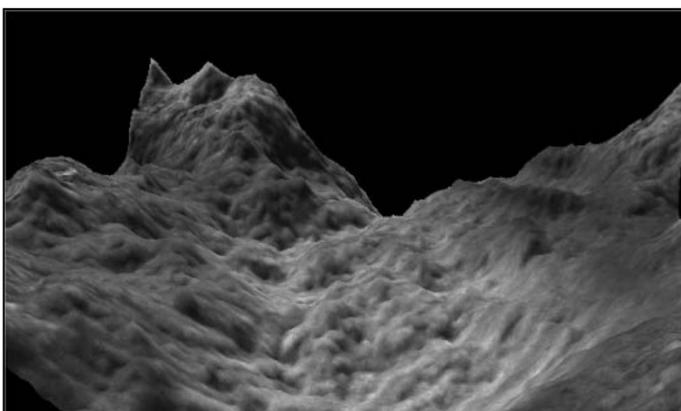


图 5.2 需要许多三角形来渲染这片混乱的地形。

78.5. CLOD 受损的 Geomipmapping

然而,并非所有算法都担心三角形分布到需要更多细节的区域。Geomipmapping 不会将更多的三角形分布到需要更多细节的区域,但 Rottger 的四叉树算法(在第 6 章,“攀爬四叉树”中)可以。

因此,说 CLOD 作为一个整体向需要它的区域添加更多细节并不总是正确的,但在大多数情况下确实如此。我是否已经彻底混淆了你?

像你以前从未剔除过的一样剔除!

基于 CLOD 的算法的另一个优点是它们允许比蛮力方法更有选择性地剔除多边形。这意味着看不到的多边形不会被发送到 API。例如,我们即将开始工作的 geomipmapping 实现使用了一系列景观补丁。如果补丁不可见,我们会一举消除潜在的 289 个渲染顶点(对于 17×17 顶点补丁)。这是显卡的巨大负载,而且剔除甚至不是 CPU 密集型的。用一个简单的方法,让 GPU 和 CPU 都满意,让你的主板整体更快乐。

不过,在 CLOD 地形的土地上,并非一切都令人愉快使用 CLOD 地形算法确实有一些缺点。

奇怪的是,今天早些时候,宇宙通过将 8 月刊的游戏开发者杂志放在我的邮箱中帮助我撰写了这一部分。大多数 CLOD 算法的主要缺点是每帧更新多边形网格所涉及的“簿记”。¹

在设计大多数这些算法(geomipmapping、Rottger 的四叉树算法和 ROAM)时,这种“记账”缺陷几乎没有那么普遍。这是因为算法希望将大部分工作负载放在 CPU 上,并向 GPU 发送尽可能少的信息。然而,从那以后,情况发生了很大变化。现在我们想把更多的注意力放在 GPU 上而不是 CPU 上。

结束你的
CLOD 地形介绍

显然,如果 geomipmapping、四叉树和 ROAM 算法已经过时,你现在就不会阅读它们,

地理映射理论79

这意味着某个地方的某个人（哎呀，甚至可能是我）为每种算法提出了一些优化，这些优化仍然使它们对现代地形渲染很重要。考虑到这一点，我将不再喋喋不休地谈论 CLOD，而是开始喋喋不休地谈论如何实施 geomipmapping。

半 CLOD 的 Geomipmapping 理论

受损

由 Willem H. de Boer 开发的 Geomipmapping 是一种 GPU 友好的 CLOD 算法。它也是一个简单的算法，非常适合您过渡到 CLOD 景观的土地。在我们进行的过程中，您可能需要参考实际的 geomipmapping 白皮书，为了方便起见，我将其放在随附的 CD-ROM 中。（它被命名为算法白皮书/geomipmapping.pdf。）

简单的基础知识如果您熟悉 mipmapping 的

纹理概念，那么 geomipmapping 对您来说应该是熟悉的基础。概念是相同的，只是我们处理的是一块地形的顶点，而不是处理纹理。geomipmapping 的驱动概念是你有一组地形。对于这个解释，我会说它是一个大小为 5 个顶点的补丁（一个 5×5 补丁）。该 5×5 补丁将具有多个详细级别，其中 0 级是最详细的，在这种情况下，2 级是最不详细的。

如果您需要直观地解释每个补丁在不同级别的外观，请查看图 5.3。在图中，黑色顶点没有发送到渲染 API，但白色顶点是。

如果您参考 Willem de Boer 的 geomipmapping 白皮书，您可能已经注意到图 5.3 中显示的三角形排列与白皮书中显示的排列略有不同。我这样做的原因稍后你会明白的，但现在，只要知道我这样做是有原因的。

现在是我们讨论 geomipmapping 的时候了。我之前已经给你介绍了基础知识，但现在是你了解一切的时候了……

80.5. CLOD 受损的 Geomipmapping

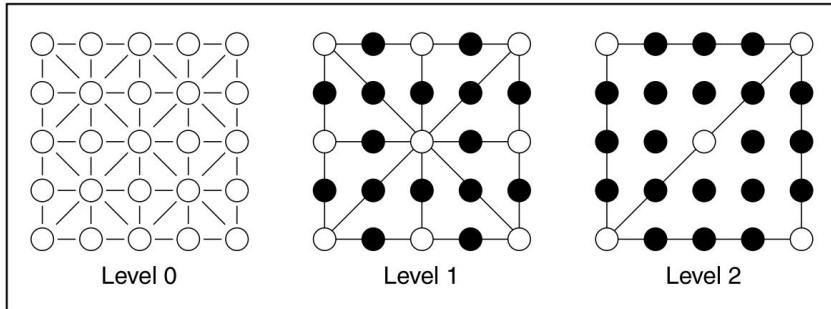


图 5.3 一块地形的三角形排列,其中最详细的排列在左边,最不详细的排列在右边。

好吧,几乎所有东西。为了保护您,我可能会保留一些信息。

正如我之前所说,geomipmapping 类似于纹理 mipmaping,只是我们使用的是土地补丁而不是纹理补丁。我们需要做的是,从用户在 3D 空间中的点 (相机的眼睛位置)开始,将查看器周围的所有块都设为最详细的,因为这些块是用户看到最多的部分。在一定距离之外,我们将切换到较低级别的补丁细节。而且,在另一个距离,我们将切换到更低级别的细节。图 5.4 直观地解释了这一点。

正如您在图中看到的,查看器位置的直接区域中的补丁的细节级别 (LOD) 为 0,这意味着这些补丁具有最高的细节级别。随着补丁变得越来越远,它们变为级别 1,这是第二高的细节级别。甚至离观察者更远,补丁的级别为 2,这是图像中呈现的最低级别的细节。

三角形排列变得简单早些时候,您可能已经注意到,我在图 5.3 中使用的三角形排列与我提到的 geomipmapping 论文中介绍的完全不同 (在 CD 上或 Internet 上的 URL 的末尾提供本章)。如果您现在无法访问该论文,请查看图 5.5 以了解该论文建议的三角形排列是什么样的。

2	2	2	2	2	2	2	2
1	1	1	1	1	1	2	2
1	1	1	1	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
Eye Position							
0	0●	0	0	1	1	2	2
0	0	0	0	1	1	2	2
1	1	1	1	1	1	2	2

图 5.4 在 geomipmapping 算法中,随着一块地形离观察者越来越远,它会切换到较低的细节层次。

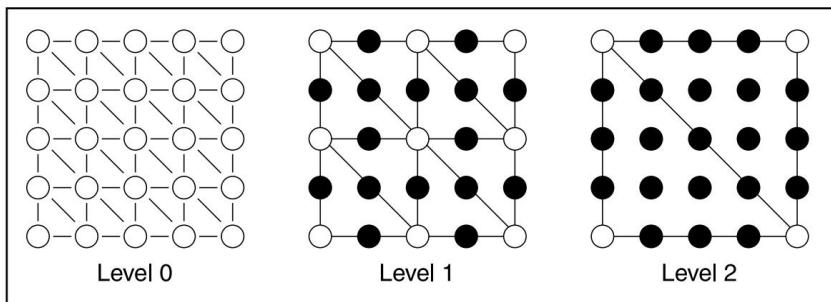


图 5.5 geomipmapping 白皮书中呈现的补丁渲染的三角形排列。

这种安排似乎是一个更好的主意,而且在很大程度上是这样。(警告:我会在这里稍微偏离正题。)

如果您打算使用顶点缓冲区来渲染补丁,三角形条绝对是要走的路,这是我建议的。但是,因为实现顶点缓冲区渲染系统高度依赖 API,所以我选择使用立即模式渲染,因为如果需要,它更容易转换为另一个 API 的语法。使用顶点缓冲区进行渲染可以大大提高任何地形实现的速度,因为它减少了发送每个顶点、纹理坐标、颜色、

82.5. CLOD 受损的 Geomipmapping

依此类推到 API 中。此外，大多数显卡更喜欢以顶点缓冲区的形式发送顶点信息。

最后，我建议您使用顶点缓冲区进行地形渲染。您将获得巨大的速度提升，而且为实现它付出额外的努力总是值得的。如果您想查看使用 Direct3D 顶点缓冲区的 geomipmapping-esque 技术示例，请查看“使用联锁瓷砖简化地形”，发表在 Game Programming Gems，第 2 卷。

无论如何，是时候回到话题了。图 5.3 所示的排列是我们渲染补丁的方式。这种安排为我们提供了一个巨大的好处：它允许我们在需要时轻松跳过渲染顶点，这很常见。这使我进入了我们的下一个讨论主题。

破解和破解，但大多只是破解当您处理 CLOD 地形算法时，您必须处理破解的主题。在 geomipmapping 的情况下，当一个高度详细的补丁位于较低详细的补丁旁边时，就会发生裂缝（见图 5.6）。

从图中可以看出，左侧的补丁比右侧的补丁具有更高的细节层次。我们的问题在于 A 点和 B 点。问题在于 A 点左侧的细节层次比 B 点的要高。这意味着左边的补丁在 A 点渲染精确的高度，但是正确的补丁只是得到它上面的高度和它下面的高度的平均值。

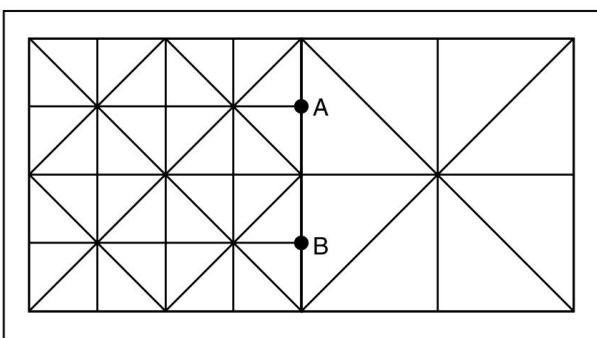


图 5.6 两个并排的补丁，具有不同的详细程度。

地理映射理论83

这整个“破解”的事情可能看起来没什么大不了的,但请查看图 5.7,它显示了我的 geomipmapping 实现的屏幕截图,没有采取非破解措施。

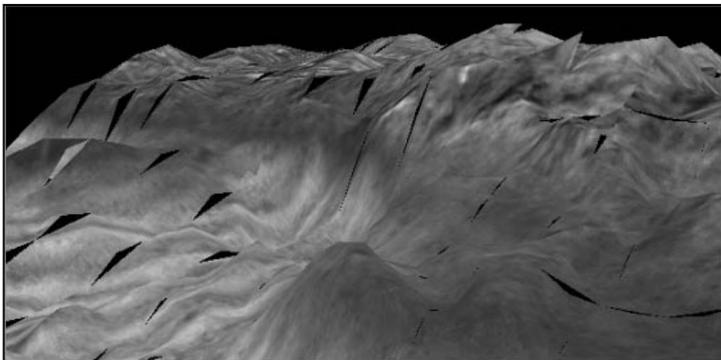


图 5.7一个 geomipmapping 实现的截图,它没有实现反破解措施。

这不是一个平坦的景观,是吗?看看风景中所有那些巨大的洞。让我们修复它!

破解您的 Geomipmapping 引擎
破解您
的 Geomipmapping 引擎比听起来容易得多。让
某人(可能是我)向您解释这个概念还有一个额外的好处,这使整个过程变得
像……嗯,简单的事情一样简单。

我们有两种可能的方法来解决开裂问题。一种方法是将顶点添加到具有较低细节量的面片,以便相关顶点与较高细节面片的相应顶点具有相同的高度。不过,这个解决方案可能很难看,这意味着我们必须重新排列补丁(添加另一个三角形扇形)。

解决这个问题的另一种方法是从更详细的补丁中省略顶点。这可以无缝且毫不费力地解决开裂问题。查看图 5.8,看看简单地省略一个顶点并修复裂缝是多么容易。

84.5. CLOD 受损的 Geomipmapping

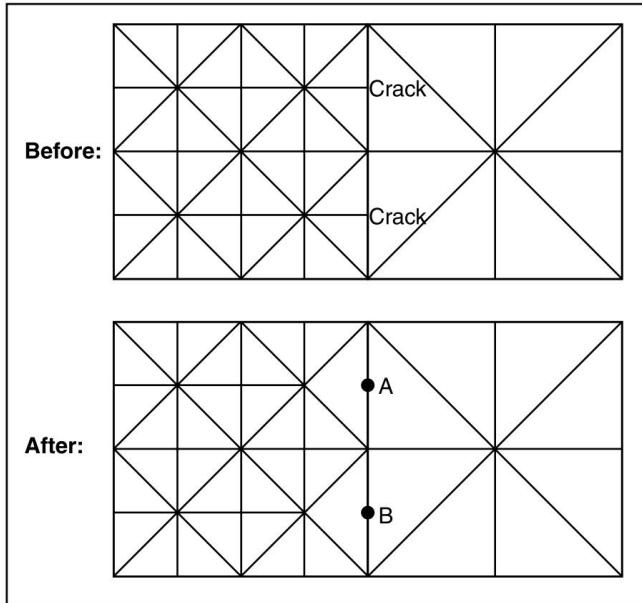


图 5.8 通过省略在点 A 和 B 处渲染顶点来消除裂纹。

你在哪里破解？

您知道导致裂缝的原因以及如何修复它们。真正的问题是：你怎么知道什么时候修复它们？基本上，当您渲染当前补丁时，您需要测试它周围的补丁（参见图 5.9）以查看它们是否具有较低的细节级别。如果是，你知道你需要省略一些顶点。

测试每个补丁并不困难。您只需要实现一系列简单的if-else语句。（伪代码如下所示。）

如果 LeftPatch.LOD 小于 CurrentPatch.LOD

 渲染左顶点=真；

别的

 渲染左顶点=假；

如果 RightPatch.LOD 小于 CurrentPatch.LOD

 渲染右顶点=真；

别的

地理映射理论85

渲染右顶点 = 假;

如果 UpperPatch.LOD 小于 CurrentPatch.LOD

 渲染上顶点=真;

别的

 渲染上顶点=假;

如果 LowerPatch.LOD 小于 CurrentPatch.LOD

 渲染下顶点=真;

别的

 渲染LowerVertex = false;

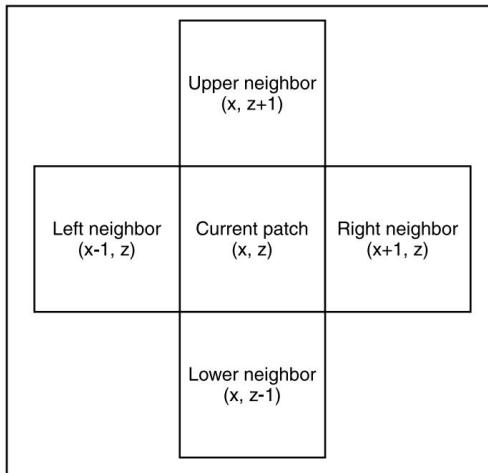


图 5.9需要测试以查看它们是否具有较低 LOD 的相邻块。

看看它有多简单?测试后,在渲染三角形扇形时,您会跳过较粗面片方向的顶点。例如,如果右侧面片的细节层次较粗,而您当前的面片细节层次较高(渲染了多列/行的三角形扇形),那么您只想跳过最右边的顶点补丁(见图 5.10)。

86.5. CLOD 受损的 Geomipmapping

警告

请注意,您只省略了必要的顶点。否则,您最终可能会得到一个充满您不想省略的顶点的补丁。例如,在图 5.10 中,补丁由多列和多行三角形扇形组成。

您不想为每个风扇省略正确的顶点;您只想在最右边的列中为粉丝省略正确的顶点。

这就是您简单的 geomipmapping 理论!现在是时候实施我们刚刚学到的所有内容了。

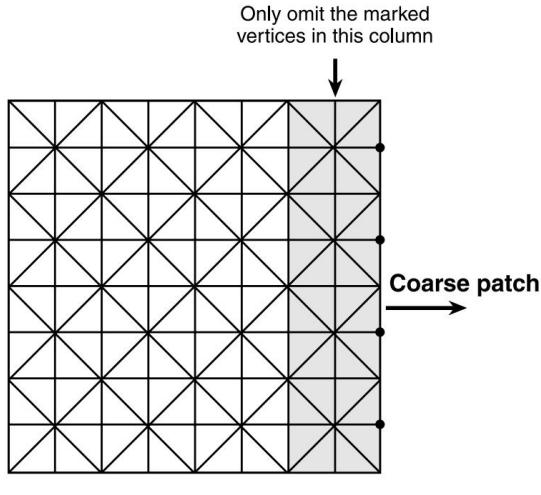


图 5.10 需要在最右侧列中省略扇形中的正确顶点以防止与右侧的补丁一起破裂的补丁。

实施

Geomipmapping 的 非常轻微的 CLOD 受损

您知道 geomipmapping 基础知识背后的理论,但现在我们需要实现它。这不应该对你的大脑造成太多负担。困难的部分已经结束,像往常一样,我们将一步一步地迈出。喝点咖啡因,锁上门,听点好听的音乐!

修补它因为 geomipmapping

由一系列补丁组成,所以通过创建补丁数据结构开始实现可能是一个好主意。结构确实不需要包含太多信息,我们需要包含的信息越少越好。事实上,这将是你在本书中看到的最小的结构。

不要太习惯它漂亮的尺寸!

补丁结构真正需要的是两个变量。一个变量将跟踪补丁的当前细节级别,一个变量将存储从补丁中心到相机位置的距离。这里的所有都是它的!这就是整个补丁数据结构。

在这里,在代码中:

```
结构 SGEOMM_PATCH
{
    浮动 m_fDistance;
    诠释 m_iLOD;
};
```

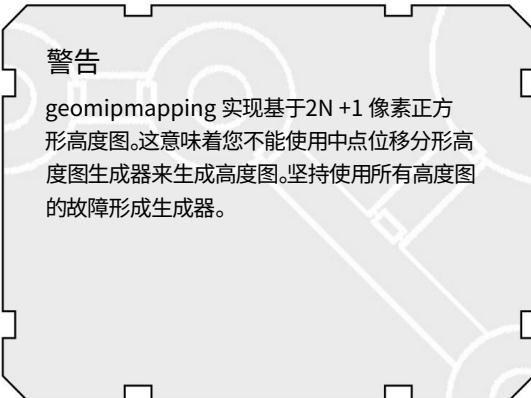
它可能看起来像一个很小的结构,但请记住:大东西装在小包装里。与该软件包一样小,我们将不断使用它,因此请确保您花费数小时来记住它的成员。

创建基本的 Geomipmapping 实现是的,不再

有这种懦弱的二元数据结构的东西了。现在我们将开始研究 geomipmapping 的主力

88.5. CLOD 受损的 Geomipmapping

引擎 geomipmapping 类。首先,我们需要一个指针来保存我们的补丁信息,它将在我们的演示中的某个时间点动态分配。接下来,我们需要计算块大小(以顶点为单位)以及地形每边有多少块。补丁大小完全由用户决定,所以我们可以让他指定补丁大小。



他喜欢初始化
班上。(我倾向于坚持
使用大约 17×17 顶点大
小的面片,因为它提供了细
节和速度的完美结合。本章
的解释假设面片大小为该大
小。)

Geomipmapping 初始化对于 geomipmapping 系统的初始
化,我们需要用户提供的只是他想要的补丁大小。(我将恢复我对 17×17 顶点的建
议。)有了它之后,我们就可以初始化系统了。

首先,我们需要计算地形每侧将有多少块。我们通过获取高度图的大小并将其除以单
个块的大小来计算这一点,如图 5.11 所示。

P 表示每边的补丁数, h 表示高度图的大小, s 表示单个补丁的大小。记住这个等式,看
看图 5.12,看看我们稍后将把哪些变量插入到等式中。

在我们计算出每边的补丁数量之后,我们需要通过对每边的补丁数量进行平方来分配地
形补丁缓冲区。

(这是我们刚刚计算完的值。)

```
m_pPatches = 新的 SGEOMM_PATCH [SQUARE(m_iNumPatchesPerSide)];
```

接下来,虽然它不是初始化的必要部分,但我想计算一个补丁可以实现的最大细节级别。
请注意,最大细节级别是最不详细的级别,即最详细的级别为 0。随着级别的增加,细节减
少。

实施 Geomipmapping 89

$$p = \frac{h}{s}$$

图 5.11 计算地形网格每一侧的补丁数量的方程。

以下是计算：

```
iDivisor= m_iPatchSize-1;
同时 (iDivisor>2)
{
    iDivisor = iDivisor>>1;
    iLOD++;
}
```

我们在这里所做的只是查看将 iDivisor 降到 2 需要多少次循环。当 iDivisor 达到 2 时，我们不能再进一步下降，我们已经计算了我们可以使用的细节级别。

对于 17×17 的补丁大小，我们的最大细节级别是 3，这意味着我们有四个不同的细节级别（0、1.2 和 3）可供任何单个补丁选择。这就是初始化！现在我们将继续讨论非常大的关闭部分。

Geomipmapping 关闭关闭 geomipmapping 系统既简单又常规。

我们需要做的就是释放我们为补丁缓冲区分配的内存并重置所有类的成员变量。

Geomipmapping 维护与我们在过去三章中一直使用的地形不同，CLOD 地形算法需要每帧更新（这就是为什么它被称为连续细节级别）。大多数基于 CLOD 的算法需要在更新阶段完成大量维护工作，但 geomipmapping 不是其中之一。我们在更新功能期间必须做的工作非常少；它只包括确定我们的补丁应该是哪个 LOD。

90 5. CLOD 受损的 Geomipmapping

对于我们geomipmapping实现的更新功能,我们需要更新每一个patch;因此,我们需要制作一对for循环:

```
for(z=0;z<m_iNumPatchesPerSide;z++)
{
    对于 (x=0;x<m_iNumPatchesPerSide;x++)
    {
}
```

我们需要在这个循环中做的第一件事是计算从观察者的位置 (相机的眼睛位置)到当前补丁中心的距离。这个计算应该在你的高中数学课上看起来很熟悉,他们将距离公式钻到你的脑海中。以防万一你像我一样在所有这些课程中都睡过觉,这里又是这样 (见图 5.12) :

$$dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

图 5.12 3D 距离公式。

记住这个等式,查看图 5.13,看看我们将把哪些变量插入到等式中。

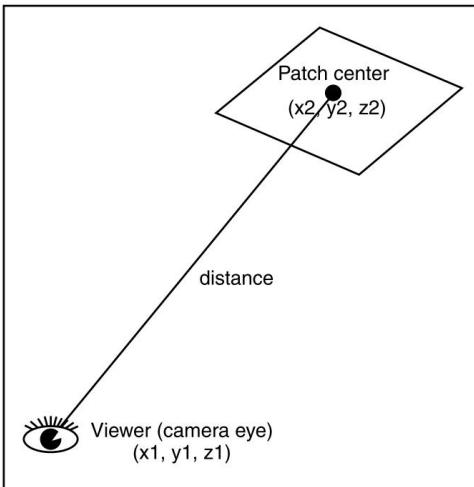


图 5.13 距离计算中涉及的变量 (从观察者到当前补丁的中心)。

实施 Geomipmapping

91

这是代码中相同距离计算：

```
m_pPatches[iPatch].m_fDistance= sqrtf( SQUARE( ( fX-
                                                 camera.m_vecEyePos[0] )+ SQUARE( ( fY-
                                                 camera.m_vecEyePos[1] )+ SQUARE( ( fZ-
                                                 camera.m_vecEyePos[2] ) ) );
```

在我们计算出与观察者的距离之后,我们可以计算出要给出补丁的详细程度。在代码中,我通过对距离进行硬编码来计算级别。(我在几段之外的代码段中执行了此操作;您可能想稍微向前浏览一下,看看我在说什么。)但是,对于您的引擎,您需要一种更严格的方法来确定什么补丁应该的详细程度。例如,在 geomipmapping 白皮书中,Willem de Boer 提出了一种屏幕像素确定算法,因此当补丁更改其细节级别时,不会出现过多的弹出。

弹出是指多边形对象更改为不同的细节级别。这种变化可能很明显,也可能不明显。例如,从 1 级补丁更改到 0 级补丁不会引起太多弹出,因为 1 级补丁仍然非常详细(至少对于 17×17 补丁)。然而,从 3 级补丁更改到 2 级补丁会导致相当多的爆裂,因为您要从 8 个三角形变为 32 个。虽然这与第一个补丁中添加的三角形比例相同,但在 3 级到 2 级的变化。任何 CLOD 算法的主要目标之一是减少甚至完全消除弹出。稍后我们将对此进行更多讨论。

无论如何,对于本书的 geomipmapping 实现,我只是对 LOD 距离变化进行硬编码。(我想把练习留给你,读者。是的,我知道我是个好人。)这是改变 LOD 的代码片段:

如果 (m_pPatches[iPatch].m_fDistance<500)

```
    m_pPatches[iPatch].m_iLOD=0;
```

否则如果 (m_pPatches[iPatch].m_fDistance<1000)

```
    m_pPatches[iPatch].m_iLOD=1;
```

否则如果 (m_pPatches[iPatch].m_fDistance<2500)

```
    m_pPatches[iPatch].m_iLOD=2;
```

92.5. CLOD 受损的 Geomipmapping

否则如果 (*m_pPatches[iPatch].m_fDistance*>=2500)

```
m_pPatches[iPatch].m_iLOD=3;
```

这些距离有效地结合了速度和细节。如果演示在您的视频卡上有点迟钝,您可能想要更改这些距离,但不要。我们将在本章后面进行一些速度优化,所以不要绝望!这就是更新 geomipmapping 补丁的全部内容……至少现在是这样。现在是时候进入任何地形实现的有趣部分了:渲染它!

Geomipmapping Rendering 这可能是本章中你会遇到的最难的部分,而且还不错。有时有点复杂,但我会逐步指导您。

稍微拆分一下 渲染我们需要渲染的所有内容的

最简单方法是稍微拆分各个渲染例程,这样代码就不会变得过于臃肿。这可能会增加一点函数开销,但只要我们聪明地做事,它就不会太糟糕。

按照我的理解,geomipmapping 类应该有一个高级的渲染功能,除了几个低级的,连续的。

例如,最高级别的渲染函数是 Render。 Render 之后是 RenderPatch ,然后是 RenderFan。 RenderVertex 位于最低级别。使用这些函数,我们稍微增加了函数开销,但我们显着减少了代码的丑陋。权衡是值得的。如果你很难掌握我打算使用的功能,请查看图 5.14。至于实现我们的渲染系统,让我们从低处开始,逐步向上。

渲染顶点函数

系统的顶点渲染功能并没有什么特别之处,但公平地说,它是一个小功能。我们会经常调用它,这使它成为一个完美的内联函数。 RenderVertex 设置顶点的颜色,该颜色基于从光照贴图中提取并乘以灯光颜色的相应 RGB 值的着色值。然后 RenderVertex 将纹理坐标发送到渲染 API (用于细节映射,如果需要,以及用于颜色纹理)。之后,您只需将缩放后的顶点坐标发送到渲染 API 而已!

实施 Geomipmapping 93

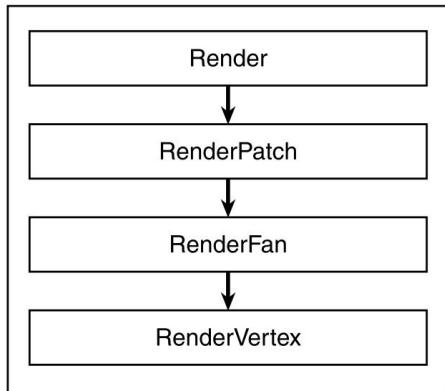


图 5.14 Geomipmapping 类的渲染架构。

RenderFan 函数

每个 geomipmapping 补丁被分解成几个粉丝,无论是 1 个补丁还是 256 个补丁。使用此函数可以清除 RenderPatch 中使用的大部分代码,这将在下一节中讨论。

补丁渲染功能所做的只是渲染单个三角形扇形。因此,函数需要能够接受作为参数的扇形中心和扇形一侧的大小,以便函数能够渲染它。 RenderFan 还需要从补丁中获取邻居信息。嗯,有点。邻居信息是针对个人粉丝的。如果由于右侧的补丁较粗,补丁需要省略一个顶点,但当前扇形在补丁的中间,则邻居结构将所有邻居显示为真。(只有补丁右边缘的扇形需要担心顶点遗漏。)但是,如果正在渲染扇形,则需要顶点遗漏。例如,如果扇形位于面片的右边缘,而当前面片右侧的面片具有较粗的 LOD,则当前扇形将需要省略右顶点的渲染。

渲染补丁函数

补丁渲染功能对整个渲染系统至关重要,因为不渲染补丁,您就看不到地形。大多数防裂步骤发生在此函数中,其余(顶点省略)发生在 RenderFan 函数中。

94.5. CLOD 受损的 Geomipmapping

还记得我在“你在哪里破解？”中向你展示的伪代码。好吧，这就是我们需要实现它的地方。我们需要填写一个“邻居结构”，它是一个简单数据结构，包含四个布尔标志，用于判断相邻补丁（左、右、上、下补丁）是否比当前补丁具有更高的细节级别。如果邻居被标记为真，那么我们可以在补丁的那一侧正常渲染，因为我们不需要采取特殊措施来防止裂缝。如果它被标记为假，那么我们确实需要采取特殊措施。

笔记

当我们在本章中谈论“更高级别的细节”时，我们指的是“更低的细节级别”。这是因为

我们的关卡系统从高（细节量低）开始，随着关卡的降低而变得更详细，以 0 级作为最详细的结束。

当我们谈论高/低级别的细节时不要感到困惑。

在防裂步骤之后，我们需要弄清楚如何开始渲染三角形风扇。这比听起来要复杂一些，但并不太难。

最困难的部分是试图弄清楚 cen 之间的距离。三角形风扇的术语。完成后，我们就像黄金一样！

我们如何计算每个风扇中心之间的距离？好吧，虽然看起来有点奇怪，但我采取的方法是从单个补丁大小开始，并试图找出将其除以得到每个风扇中心之间的长度。我从一个补丁大小的除数开始，然后做了一个 while 循环来计算总补丁大小除以多少。例如，如果补丁是 0 级，我们将单独的补丁大小除以自身，这会在我们渲染的每个扇形之间产生 1 个单位的长度。（在我们深入到 RenderVertex 函数之前，我们不想缩放顶点。）对于级别 1 的补丁，每个扇形之间的距离将是 2 个单位，对于级别 2 将是 8 个单位，依此类推。前面计算的代码如下所示：

```
fDivisor = ( 浮动 )m_iPatchSize;
fSize = ( 浮动 )m_iPatchSize; =
iLOD      m_pPatches[iPatch].m_iLOD;
```

实施 Geomipmapping 95

```
//找出我们将有多少个粉丝部门
```

```
而 (iLOD>=0)
```

```
iDivisor = fDivisor/2.0f;
```

```
//每个三角形扇形中心之间的大小
```

```
fSize/= fDivisor;
```

不过,这里的计算并不完全正确。当我们使用它们时,它们会产生如图 5.15 所示的地形。

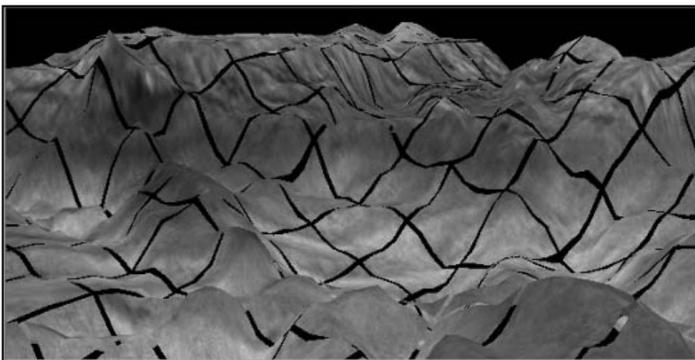


图 5.15哎呀!似乎执行了一些错误的计算!

这些计算出了什么问题?好吧,我们做错了一件简单的事情。我们希望除数变量 `fDivisor` 是 2 的幂。还记得当除数等于补丁大小时,0 级补丁的粉丝之间的距离是 1 个单位吗?嗯,从一个中心到另一个中心,我们至少需要一个 2 个单位的间隔 (见图 5.16)。

您看到图 5.16 中顶部的一对风扇如何以一个单位的间隔相互重叠 (产生相当丑陋的结果),以及底部的两个风扇如何以两个单位的间隔完美地组合在一起?

好吧,我们需要更改之前的风扇中心间隔计算,使补丁之间的最小间隔为 2 个单位。你问我们怎么做?这很简单。我们只是将初始除数变量设置为补丁大小减 1,这总是使除数变量为 2 的幂,从而解决了我们之前遇到的所有问题。查看新代码。(`fDivisor` 变成了 `iDivisor`,这样我们可以加快计算速度。)

96.5. CLOD 受损的 Geomipmapping

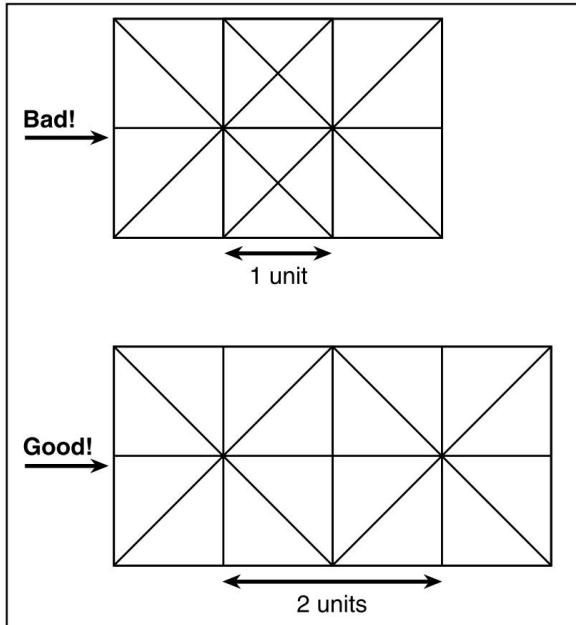


图 5.16 风扇中心之间的一个单位间隔与补丁中心之间的两个单位间隔。

```
fSize = ( 浮动 )m_iPatchSize;
iDivisor= m_iPatchSize-1;
iLOD      = m_pPatches[iPatch].m_iLOD;
```

//找出我们将有多少个粉丝部门

而 (iLOD>=0)

```
iDivisor = iDivisor>>1;
```

//每个三角形扇形中心之间的大小

```
fSize/= iDivisor;
```

完成之后,渲染补丁的三角扇就变得微不足道了。

您只需要确保从fSize的一半开始渲染,因为那是第一个风扇的中心所在的位置。我们需要检查

每个扇形是否需要省略顶点。这意味着我们需要使用

我们之前填写的补丁邻居结构的信息,并将其应用于正在渲染的每个补丁边缘的风扇,如下所示:

实施 Geomipmapping 97

//如果这个风扇在左行,我们可能需要调整它的渲染以防止出现裂缝 if(x==fHalfSize)

```
fanNeighbor.m_bLeft=patchNeighbor.m_bLeft;
别的
```

```
fanNeighbor.m_bLeft=真;
```

//如果这个风扇在底行,我们可能需要调整它的渲染以防止出现裂缝 if(z==fHalfSize)

```
fanNeighbor.m_bDown=patchNeighbor.m_bDown;
别的
```

```
fanNeighbor.m_bDown=真;
```

//如果这个风扇在右排,我们可能需要调整它的渲染以防止出现裂缝

```
if( x>=( m_iPatchSize-fHalfSize ) ) fanNeighbor.m_bRight=
patchNeighbor.m_bRight;
```

别的

```
fanNeighbor.m_bRight=真;
```

//如果这个风扇在顶行,我们可能需要调整它的渲染以防止出现裂缝

```
if( z>=( m_iPatchSize-fHalfSize ) )
```

```
fanNeighbor.m_bUp=patchNeighbor.m_bUp;
```

别的

```
fanNeighbor.m_bUp=真;
```

//渲染三角形扇形

```
RenderFan( ( PX*m_iPatchSize )+x, ( PZ*m_iPatchSize )+z,
fSize, fanNeighbor, bMultiTex, bDetail );
```

通过填写一个单独的风扇邻居结构,我们不必继续重做补丁的邻居结构。然后将扇形邻居结构发送到扇形渲染函数,在那里它用于找出是否需要从渲染中省略任何顶点。这就是所有低级渲染功能。现在我们需要简要讨论该类的高级 Render 函数,这是用户将使用的函数。我们可以在世界上释放 demo5_1!

98.5. CLOD 受损的 Geomipmapping

渲染函数

好吧,我们刚刚完成了简单的 geomipmapping 实现。你开始兴奋了吗?我知道我是!

在高级渲染函数中,我们需要遍历所有补丁并调用 RenderPatch 函数,但我们将有三个不同的补丁渲染循环。还记得我们的蛮力实现吗?如果用户启用了多重纹理,我们只需要制作一个渲染循环;然而,如果用户没有启用多重纹理,我们可能需要制作一个纹理贴图通道和一个细节贴图通道。但是,地形实现的两个不同渲染通道绝不是一件好事,因此如果用户没有多重纹理支持,一个选择是避免细节映射通道。如果您从头开始连续阅读本书的章节,您应该已经熟悉了这个概念。

您唯一需要做的就是遍历所有补丁并使用 RenderPatch 函数来渲染它们。

而已!我们已经完成了基本的 geomipmapping 实现。查看 demo5_1 (在 CD 上的 Code\Chapter 5\demo5_1 下) 并查看图 5.17 中该演示的示例屏幕截图。它在左侧显示纹理/细节映射图像,在右侧显示同一图像的线框。请注意线框图的一半,您附近的补丁如何更详细,而更远的地方则不那么详细。这就是 CLOD 算法的美妙之处!

存在待修复的问题

是的,是的,我知道。我们刚刚完成的 geomipmapping 实现存在一些问题。例如,除非您使用的是真正高端的显卡,否则您可能在之前的演示中遇到了非常糟糕的帧速率。(我使用的是 GeForce 4 Ti4600,这是撰写本文时市场上最好的显卡,在演示中我获得了稳定的每秒 45-50 帧。)该演示还遇到了一些地形补丁时的爆裂声改变了他们的详细程度。我们将在接下来的部分中解决所有这些问题以及更多问题,所以不用担心!

为引擎添加一点新鲜感首先,我认为我们应该加快实现速度。加快速度很容易。它只涉及进行一些截锥体剔除。和

实施 Geomipmapping 99

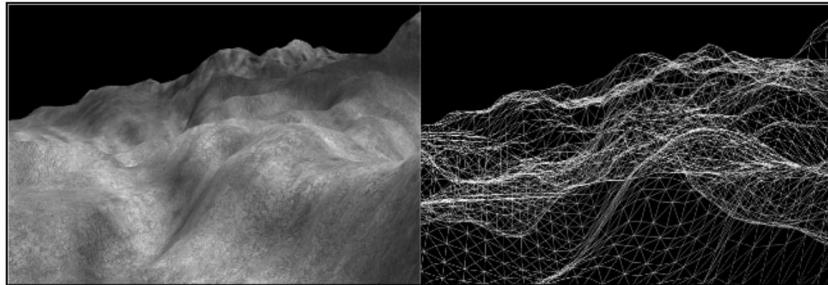


图 5.17 纹理和细节映射的地形截图（左）及其对应的线框图像（右）, 取自 demo5_1。

因为我们的地形被分成了几个块, 我们最好的办法是根据视锥剔除这些块, 这意味着不需要做很多测试。

像你以前从未剔除过的一样剔除...再次, 我们将在这里进行一些基本的平截头体剔除。我根据 Mark Morley 的文章 “OpenGL 中的截锥体剔除”中的一些信息向CCAMERA类添加了一些截锥体计算功能, 我认为这是我在 Internet 上看到的最好的截锥体剔除教程。 (你可以在 <http://www.markmorley.com/opengl/frustumculling.html> 找到它。) 是的, 我承认, 我的数学知识不是太丰富 (请注意本书中缺乏复杂的数学!), 但这可能是一件好事 除非你是一个巨大的数学迷, 在这种情况下, 我可能会收到一些仇恨邮件。

无论如何, 这里是我们将要做的基础知识。我们将根据视锥剔除一块地形 (图 5.18, 以防你需要视觉更新), 以便我们消除任何最终浪费的额外 CPU/GPU 工作。 (如果观众看不到它, 那么渲染/更新它就没有意义了。)

我们需要针对该截锥体测试一片地形。为此, 我们从补丁中制作了一个 Axis-Aligned Bounding Box (AABB)。

(实际上, 我们沿着立方体的线条制作了更多内容。) 然后我们想针对视锥进行测试。为了计算补丁的尺寸, 我们采用中心和一个缩放的尺寸变量 (见图 5.19)。

100 5. CLOD 受损的 Geomipmapping

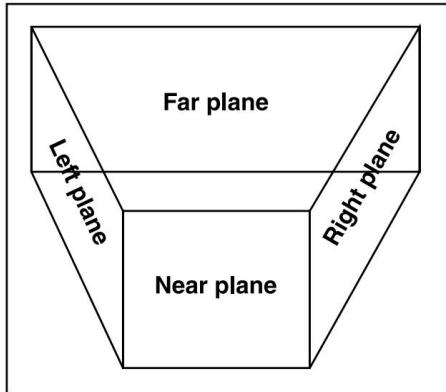


图 5.18 视锥。

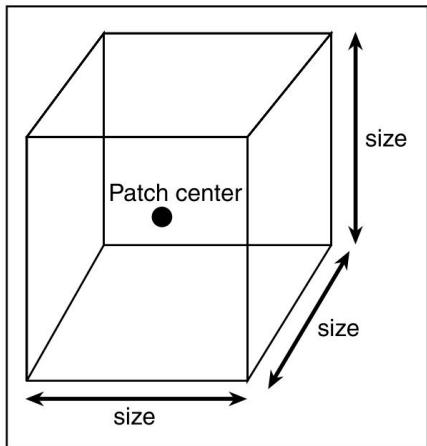


图 5.19 用 geomipmapping 补丁制作一个立方体来测试视锥。

因为我们在这里只处理补丁中心,所以你需要取作为函数参数传递的一半大小 (到 Cube Frustum 交点),并据此计算出立方体的角。我们也可以得到一个更精确的盒子,但在我的实验中

对于剔除,额外的“空间缓冲区”是必要的,这样观众就不会看到地形中的不一致(例如稍微可见,但最终还是被剔除)。

实施 Geomipmapping 101

现在您对剔除及其在地形中的实际用途有了更多的了解,请查看 demo5_2 (在 CD 上的 Code\Chapter 5\demo5_2 下) ,并亲眼见证事情变得多么快。例如,在图 5.20 中,在大约 910 个补丁中,我们只看到 369 个。

对于演示,我获得了稳定的帧速率 (80–120fps),这是使用的高度图是 demo5_1 中的高度图的两倍。

不太寒酸!

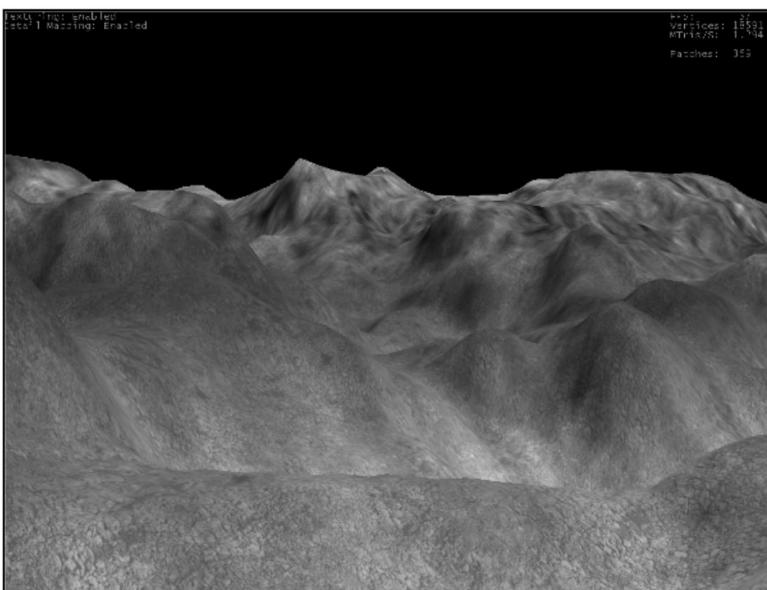


图 5.20 来自 demo5_2 的屏幕截图。

Pop Today, Gone Tomorrow 我们要解决的下一个问题比上一个问题更复杂。我们的目标是减少甚至更好地消除爆裂声。

有几种方法可以解决这个问题,我将介绍其中两种方法背后的理论。不过,这些解决方案的实际实施取决于您。

变形它!并且变形它很好!

弹出问题的第一个解决方案称为 geomorphing。

虽然这个名字听起来像是你在动漫中听到的关于巨型机甲的东西,但它与巨人绝对无关

102 5. CLOD 受损的 Geomipmapping

具有大规模杀伤能力的多层机器人。Geomorphing实际上是在一组多边形（如我们的geomipmapping补丁）中逐渐变形顶点的过程，这些多边形正在改变它们的细节级别。我个人认为机械的东西听起来更有趣，但变形的实际含义要有用得多。

你问为什么这有用？实际上，这很简单。你看，当一个补丁的细节层次很低时，它近似于几个区域的高度（见图 5.21）。并且，在图中，补丁近似值的区域用黑点标记。

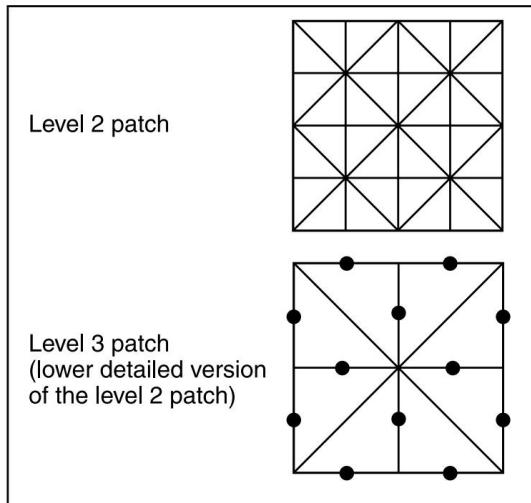


图 5.21 低层补丁如何近似低层补丁的几个顶点的高度值。

你看到图像下方的补丁上有多少个黑点了吗？

嗯，这就是低细节贴片近似的高度值。近似值接近实际值，但还不够接近。因此，当 3 级补丁细分为 2 级补丁时（反之亦然），就会出现弹出；那是因为那些近似值被真实值“替换”（或者真实值被近似值替换）。

Geomorphing 通过计算出近似值与真实值的距离并在大约 255 个步骤中将其从近似值内插到真实值来解决这个问题。（反之亦然……再次。好吧，从现在开始，我要

实施 Geomipmapping 103

假设我们正在细分补丁。如果要合并补丁，只需反转所有计算。)要计算每一步需要移动的单位数，只需使用以下等式：

$$\text{geo} = \frac{(\text{to}-\text{from})}{\text{numSteps}}$$

图 5.22 计算变形值的公式。

在等式中，“to”是您要达到的实际值，“from”是您要从的近似值，“numSteps”是您希望进行变形的步数，我建议为 255。整个过程在图 5.23 中直观地解释。

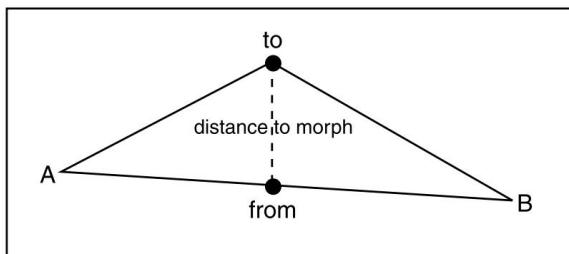


图 5.23 变形解释。

嗯，就是这样。Geomorphing 是一个简单的概念，您可以通过几种不同的方式来实现它。（我不想限制你的想象力，这就是我自己没有实现它的原因）。玩得开心，但请确保您回来阅读下一节，这有助于进一步减少地形实现中的弹出。

灯光应该发出爆裂声吗？

答案是否定的，尽管现在这并不重要。重要的是我们在地形实现中减少了一些弹出。现在，虽然在更改 LOD 级别时发生的大部分爆裂确实可以通过变形来修复，但另一个问题是正在使用 gouraud (每顶点) 光照！尽管这听起来可能没什么大不了的，但确实如此。每次 LOD 切换时，

104.5. CLOD 受损的 Geomipmapping

应用于补丁的照明变得更多/更少细节,这不应该发生。幸运的是,解决这个问题很简单,您可以通过几种不同的方式来处理它。

解决此问题的一种方法是将地形的光照贴图与地形的纹理贴图集成。这允许您在一个渲染过程中进行照明和纹理处理。不幸的是,这意味着您必须使光照贴图和纹理贴图具有相同的分辨率才能获得最佳效果。因此,如果你想动态改变光照,你也必须动态改变纹理贴图,这是相当昂贵的。

解决此问题的另一种方法是制作单独的光照贴图纹理通道。这会将光照贴图视为纹理,消除逐顶点光照,并允许您动态更新它而无需打扰纹理贴图。不幸的是,这意味着您的用户至少需要 3 个多重纹理单元才能使这种方法有效。此外,即使您的用户有 2 个纹理单元,执行 2 个渲染通道也不是一个好主意。多通道渲染对于小型模型来说没什么大不了的,但是对于像地形网格这样的巨大的多边形数据集来说,它就很重要了。

您采用的解决方案完全取决于您。如果您不需要进行动态坡度照明或光照贴图,则第一种方法非常好,如果您知道您的用户拥有支持 3 个或更多纹理单元的图形卡,则第二种方法非常好。谨慎使用额外的纹理单元。

总结嗯,这是一个很长的章节,

但你做到了。恭喜!在本章中,您学习了所有关于一般 CLOD 地形的知识,还学习了 geomipmapping CLOD 地形算法。您学习了如何加快 geomipmapping 以及如何减少由 LOD 变化引起的弹出。在阅读接下来的几章时,请牢记所有这些技巧。振作起来,接下来是基于四叉树结构的 CLOD 算法。

参考

1 de Boer, Willem H. “使用几何 Mipmap 进行快速地形渲染”。2000 年 10 月。<http://www.flipcode.com/tutorials/geomipmaps.pdf>。

第 6 章

攀登

这

四叉树

106 6. 攀登四叉树

我们的 CLOD 算法已经完成了三分之二。在本章中，您将了解 Stefan Roettger 的四叉树地形算法。这个算法非常酷而且非常快。既然您已经阅读了第 5 章“为 CLOD 受损的地理映射”，那么四叉树地形算法应该不会太难掌握。当你了解了一种 CLOD 算法后，学习新的 CLOD 算法的难度就大大降低了。

本章的议程如下所示：

- 解释什么是四叉树 ■ 典型四叉树结构在地形中的应用
- 四叉树算法的实现

所以，事不宜迟……等一下 我已经用过那个了。
嗯……让我们开始这一章。

四边形确实长在树上

谁告诉你钱不会长在树上的人是可悲的错误。
相比之下，四边形确实有在树上生长的趋势。这些树被恰当地命名为四叉树，这就是我们将在下一节以及之后的一节以及整个章节中讨论的内容。

四叉树是一种二维数据结构，非常适合地形。是的，我确实说的是 2D 而不是 3D，这可能会让您开始摸不着头脑，为什么 2D 数据结构对 3D 地形引擎有好处。

好吧，您需要记住的是地形在很大程度上是一个 2D 对象。
注意我们加载的高度图不是 3D 数据集；相反，它们是 2D 数据集，而我们的地形脱离了这些 2D 数据集。
四叉树非常适合管理地形引擎的多边形。

您可能会问自己，“四叉树到底是什么？”嗯，很简单，真的。想象一个简单的四边形，其中有一个点，如图 6.1 所示。

四头肌确实长在树上107

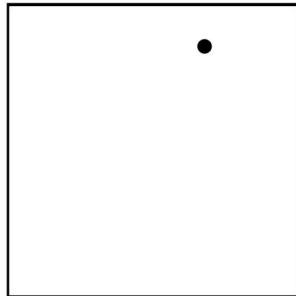


图 6.1一个简单的四边形,里面有一个点。

现在想象四边形充满了物体。您要做的就是专注于这一点;你可以不关心其他任何事情。

如果是这种情况,您不希望将点与所有其他对象组合在一起。您希望将其纳入您可能的最小组中。因此,您将这个小四边形细分一层,如图 6.2 所示。

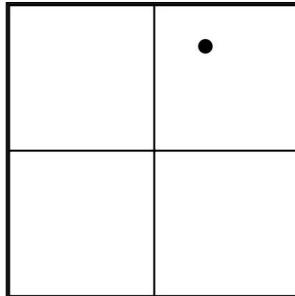


图 6.2一个细分的四边形,其中有一个点。

这稍微好一点,但是现在我们已经消除了 $3/4$ 的父四边形,我们需要更深入地关注剩余的节点。这次我们只想细分右上角的节点,如图 6.3 所示。

现在我们到了某个地方!不过,我们可以再使用一个细分。记住:越详细越好,不管别人告诉过你什么。图 6.4 显示了我们的小四细分战的最后一步。

108 6. 攀登四叉树

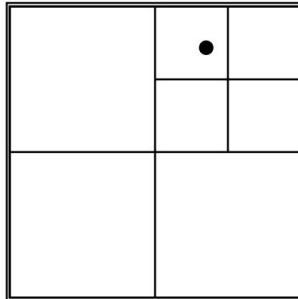


图 6.3 现在四边形正在到达某个地方，在我们想要的区域周围有相当程度的细节。

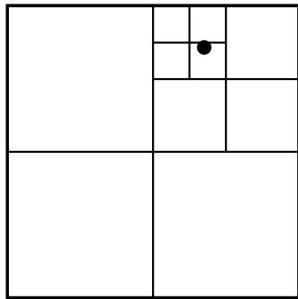


图 6.4 我们寻求……细分的最终细分级别。

这很好,但有什么意义呢?好吧,重点是:您只是将四叉树细分为该点所需的必要细节级别。我们刚刚做的就是我们将在本章中做的事情。我们将从一个简单的四边形开始,如图 6.1 所示,并以一个完全三角形的地形景观结束。当我们谈论与地形相关的四叉树结构时,不要忘记我刚刚给您的简单解释。都是同一个概念 只是应用不同!

在四边形之外思考!

我们的四叉树解释和实现基于 Stefan Roettger 撰写的一篇优秀论文,“Real-Time Generation of Continuous Levels of Details for Height Fields.1”会议论文

(发表于 WSCG98 论文集)是对 Roettger 算法的详细分析,绝对是必读的。事实上,我建议您现在就阅读它,或者通过查看 Web 上的论文(链接在本章末尾)或简单地查看我放在本书随附 CD 上的副本(算法白皮书\quadtree.pdf)。

简单的基础知识……再次讨论太多,很难知道从哪里开始。好吧,开始总是美好的。我将把本章分成三个部分,其中第一部分“四边形不会在树上生长”已经介绍了。下一部分是您当前正在阅读的部分,它将分三个不同阶段介绍四叉树算法背后的理论。然后本章的最后一节将介绍实现。话虽如此,让我们继续基本理论

部分。

矩阵中的生活……

四叉树矩阵四叉树算法完全脱离底层四叉树矩

阵。这个矩阵匹配它运行的高度图的大小,并包含四叉树中每个可能节点的布尔值。例如,与白皮书类似,让我们将矩阵作为 9×9 高度图的基础,如图 6.5 所示。

?	?	?	?	?	?	?	?	?
?	1	?	1	?	0	?	1	?
?	?	1	?	?	?	1	?	?
?	0	?	0	?	0	?	0	?
?	?	?	?	1	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	0	?	?	?	0	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?

图 6.5 用于四叉树算法的示例布尔“四元矩
阵”。

110

6. 爬四叉树

它是一堆小 0、1 和问号，但这一切是什么
 意思是？好吧，简单地说，所有这些字符都代表节点中心。
 每个 0 和 1 代表一个中心是启用 (1) 还是禁用 (0)。问号代表没有考虑的中心，这意味着我们没有细分到足够多的地方

四叉树来“关心”这些中心。我敢打赌，您是在问自己：“镶嵌后会是什么样子？”好吧，让我展示一下

图 6.6 中的你。如果你没有问自己这个问题，
 我只是继续假装你做了。

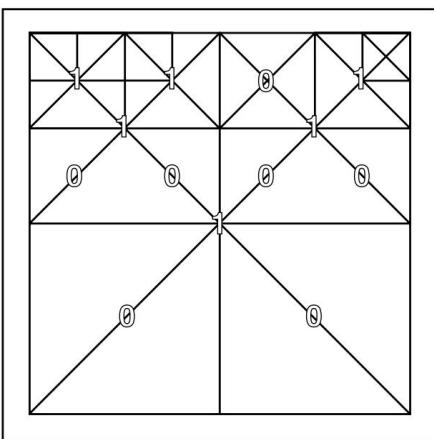


图 6.6 图 6.5 中显示的四叉树矩阵的示例曲面细分。

从图中可以看出，tessellation 正好对应四叉树矩阵。然而，矩阵没有显示的是如何镶嵌网格中的多边形需要自行调整以防止开裂。我们稍后再谈。无论如何，与以前的
 的
 记住解释，让我们继续。

查看矩阵
 环保时尚

我们将以您可能会或
 可能不熟悉。我们需要从渲染根开始
 四节点。然后我们将递归地渲染它的孩子的
 节点，然后那些子节点将渲染它们的子节点

节点。在我们到达叶节点（没有子节点的四边形节点）后，我们会将其呈现为完整或部分三角形扇形。渲染是多余的，功能是它会多次调用自己。（有点自私像那样。）在图 6.7 中，我们将通过拆开图 6.6 并查看如何渲染该网格来检查渲染过程。

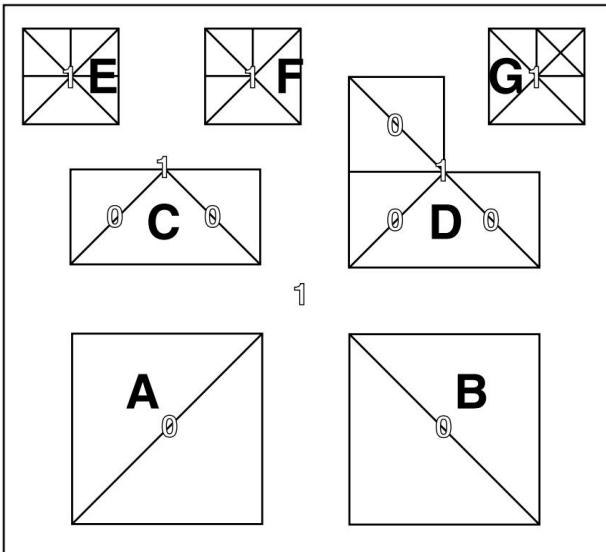


图 6.7 为我们的渲染目的从图 6.6 中挑选出网格。

对于渲染，我们必须对节点渲染进行七次调用函数（我们想一次渲染一个节点），每个字母节点一次。为了防止开裂，我们将不得不处理我们在第 5 章中谈到的著名的顶点省略技术。事实上，与第 5 章相比并没有真正改变，因为我们仍在处理三角扇，所以当我们需要防止开裂时，我们只需省略渲染单个顶点并继续使用扇形。与之前的算法类似，我们将始终检查以查看相邻节点的详细程度是否低于当前节点。如果是，我们有一些顶点遗漏要做。这些都不应该对你来说似乎太陌生了，但重要的是要记住，这种技术只有在相邻节点的细节级别没有差异的情况下才有效超过一级。

必须保持他们的生成！

对于那些知道后代的歌曲“Gotta Keep Em Separated”的人来说,这个部分的标题很好用。对于那些不这样做的人,嗯……你错过了。在渲染场景之前,我们需要生成网格。与第 5 章讨论的 geomipmapping 算法不同,该算法需要更多的更新。在每一帧中,我们需要从四叉树的高处(在父节点)开始,沿着树向下工作,看看我们是否需要更多细节,更少细节等等。这称为自上而下的方法。有道理,不是吗?

我们在“细化”步骤中要做的(我们不能真正称之为“更新”,因为我们每帧都从头开始)是测试每个节点,看看它是否可以细分。然后我们将结果存储在四叉树矩阵中。如果节点可以细分,并且我们还没有达到最大细节级别,那么我们想要进一步向下递归树并对所有节点的子节点进行相同的测试。

我们需要对整个“细分测试”进行一些扩展。我们需要讨论一些要求。首先,就像任何好的 CLOD 算法一样,我们要确保我们的网格的细节随着相机的眼睛越来越远而减少。这可以通过使用图 6.8 中所示的公式来确保,该公式可在 Roettger 的白皮书中找到:

$$\frac{1}{d} < C$$

图 6.8 全局分辨率标准方程,它确保四叉树网格是正确的细节层次。

在等式中, l 是当前节点中心到相机眼睛的距离, d 是当前节点边缘的长度, C 是控制网格整体细节水平的常数。我们将测量从节点中心到相机的距离,这与我们在第 5 章中所做的方式有点不同。这一次,我们将使用称为 L1 -Norm 的东西来计算距离。它是我们在第 5 章中使用的距离公式的更快版本,但它是线性的,而不是像我们之前使用的 L2 -Norm 那样的二次方。请参见图 6.9。

$$d = |x_2 - x_1| + |y_2 - y_1| + |z_2 - z_1|$$

图6.9 L1-Norm 距离计算方程。

你看,它与第 5 章中的距离方程并没有太大区别。唯一真正的区别是我们不需要对单个分量计算进行平方,而且我们省去了平方根的麻烦。除此之外,没有太大变化。

无论如何,前面的方程所做的只是平衡整个地形网格中可见细节的总量。常数 C 允许我们控制网格的最小分辨率。Roettger 在会议论文中建议我们将其作为实现的基础,该值默认设置为 9。但是,需要注意的是,随着 C 的增加,每帧的顶点总数呈二次方增长。

完成所有这些之后,我们需要计算白皮书所说的 f,这是确定是否细分节点的最后一步。

我们将为基本实现 (稍后的 demo6_1) 计算 f 的方式与在实现 (demo6_2.) 中为第二个 “go” 计算 f 的方式略有不同。所以,对于我们现在讨论的基本理论,我将专注于简化计算,如图 6.10 所示。

$$f = \frac{1}{d \cdot C \cdot \text{MAX}(c, 1)}$$

图 6.10 计算 f 的早期方程。

你应该已经知道 l, d 和 C 是什么,但你不知道 c 是什么。好的,我会告诉你的。c 与 C 相关,但 C 是最小全局分辨率常数,而 c 是所需的分辨率常数。

然而,真正的问题是 f 与任何事物有什么关系。嗯,f 是决定我们是否要细分节点的最后一步。在计算出它的值之后,您需要测试一件简单的事情:

如果 $f < 1$

细分节点

然后,您将true存储在当前节点的四叉树矩阵中,并继续细化节点的子节点。这就是基本的四叉树理论。让我们继续下一节。

Propagation Propaganda这个理论部分是关于如何改进你的四叉树实现的镶嵌。在上一节中,我们讨论了一般曲面细分。尽管一般的细分工作得很好,但四叉树实现的能力远不止“正常”细分。我们将讨论如何在需要的区域向我们的网格添加更多细节。例如,我们可能希望一块尖尖的地形比普通的平坦土地有更多的细节。但是,这样做需要我们的部分做一些额外的工作,所以如果您认为不需要额外的细节,请随时跳到下一部分。

我们将为每个节点预处理一个名为 d2 的值。

然后,我们将在四边形矩阵中存储一个传播值,我们希望对其进行更改以使每个条目成为一个字节而不是简单的布尔标志,以供以后使用(在镶嵌期间)。我们将为每个节点计算五个 d2 值,我们将找到这五个中的最大值。查看图 6.11 以了解需要计算 d2 值的位置。(在这些地方放置一个 X。)

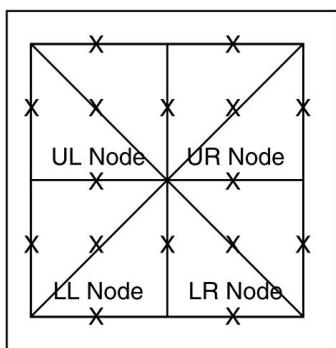


图 6.11 一个 X 标记了 d2 计算点!

对于任何 d2 计算,您需要做的就是计算 X 所在的两个顶点之间的平均值(因为这是查看者看到的当前近似高度)。然后您需要找到该值与该点的真实高度之间的绝对差。那是你的 d2 值。

接下来,我们需要采取防裂预防措施。在这一步中,我们要确保周围节点的差异不会超过一个细节级别。为此,我们将做一些与算法的其余部分不同的事情。我们不会采用自上而下的方法,而是采用自下而上的方法,从最低级别的细节开始,逐步向上。我们将计算最低节点的表面粗糙度和 d2 值,存储信息并向上传播四叉树。

笔记

d2 传播值应缩放到 [0, 1] 的范围,然后设置为对字节更友好的 [0, 255] 范围。

通过以自下而上的方式预处理四叉树,节点之间的细节层次差异

保持在小于或等于 1 的范围内。这样可以确保仅通过跳过必要的顶点就可以避免裂缝。如果有

细节级别的差异大于 1,可以简单地调整较粗节点的 d2 值以符合较低的详细节点。

以前所未有的方式剔除……我再次保证下一章的章节标题将更具原创性。无论如何,本节将相当短。

您知道平截头体剔除的基础知识,并且在先前算法中的剔除和本算法中的剔除之间的唯一变化是我们将如何处理事情。你看,在之前的算法中,我们必须每帧剔除每个补丁。当我们剔除一个四叉树节点时,我们可能会消除一整套其他节点。例如,查看图 6.12。

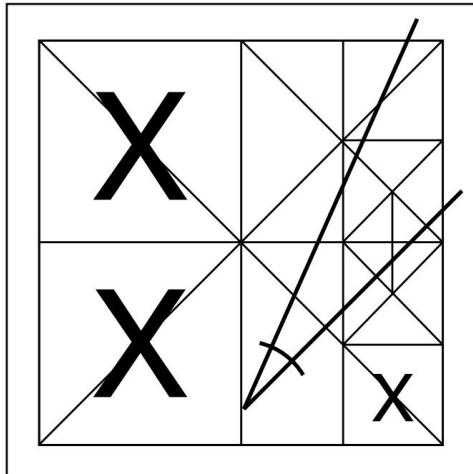


图 6.12 样本四叉树地形补丁的截锥体剔除。

如您所见，相机眼睛左侧的两个大节点被完全剔除，甚至没有费心去测试他们的孩子。在较大的高度图上，这可以节省大量的处理时间。它允许 GPU 获取更多信息，这最终是我们想要的。然而，除了前面的剔除概念之外，一切都与第 5 章中的相同。本章的理论部分到此结束。是时候着手实施了。

拥抱四叉树，
喜欢四叉树，
成为四叉树

在花费了过去几页关于 Roettger 四叉树算法背后的理论之后，我们终于要实现它了。本节仅讨论实现，因此不会讨论理论。如果您忘记了为什么某段代码会以这种方式工作，请参考理论部分的解释。

实现基础要从这个实现开始，最好创建 CQUADTREE 类。我们不要浪费时间！

上课！

在大多数情况下,该类相当简单,由保持代码整洁的“帮助”内联函数、三个成员变量和五个主要函数组成。该类的成员变量包括一个保存四叉树矩阵的布尔缓冲区、一个表示最小全局分辨率的变量和所需的全局分辨率变量。

就成员函数而言,我们需要现在标志性的Init、Shutdown、Update和Render函数,但我们还需要两个节点操作函数:RefineNode和RenderNode。让我们开始编码!

四叉树初始化和关机初始化函数是标准的,没什么特别的。我们真正需要做的就是为四叉树缓冲区分配内存并将其所有内容设置为true。这就是初始化函数的全部内容。我告诉过你这没什么特别的。然而,记住将整个矩阵初始化为真是非常重要的;如果你不这样做,你会得到一些令人讨厌的结果。

四叉树系统的关闭也非常简单。只需释放您为四叉树矩阵分配的内存即可。

四叉树更新和渲染这些高级类函数非常简单。对于该类的Update函数,您只需对RefineNode进行一次调用。因为这个函数是递归的,所以你只需要通过优化父节点来开始。通过父母的中心和它的大小,你就设置好了!

渲染函数类似于第5章的高级渲染函数。不同之处在于,您只需调用一次RenderNode,而不是循环遍历每个补丁并渲染它。与其精炼的兄弟类似,这也是一個递归函数,因此通过渲染父节点,您可以开始向下递归四叉树。渲染函数需要做的就是发送正确的纹理/细节贴图信息,以便低级RenderVertex函数(第5章中的相同函数)知道该做什么。

精炼四节点现在我们正在谈论!在这里,

您将学习如何细化单个四边形节点（尽管该函数是递归的,因此您实际上是在细化几个四边形节点）。如果您阅读本章前面的理论部分,您已经了解了如何精炼四边形节点所需的大部分知识。你确实读过那部分,不是吗?

我们将通过计算节点中心和相机眼睛位置之间的距离来开始精炼过程,使用L1

我们之前讨论过的规范。

```
distance= ( float )( fabs( cameraEye[0]-x )+ ( fabs(cameraEye [1]-
GetQuadMatrixData( iX+1, iZ )+ ( fabs(cameraEye [2]-z ) );
```

通过计算,我们可以继续计算 f,如本章前面所示。计算完后,我们可以确定是否可以细分当前节点:

```
f= fViewDistance/( ( 浮动 )iEdgeLength*m_fMinResolution*
MAX(m_fDesiredResolution*GetQuadMatrixData(iX-1, iZ)/3, 1.0f));
```

我们可以通过查看f是否小于1来了解对当前节点的细分。如果是,则可以对当前节点进行细分;如果不是,它是一个叶节点。

```
如果 (f<1.0f)
    b细分=真;
别的
    b细分=假;
```

```
//存储当前节点是否被细分 m_bpQuadMtrr[GetMatrixIndex( iX, iZ )]= bSubdivide;
```

如果该节点被细分,则它不是叶节点（显然）,这意味着它有需要进一步细化的子节点。此规则的例外情况是当前节点的边长为三个单位。如果它的边长为 3 个单位,则它是四叉树上可能的最低节点,我们不能在树上进一步递归。

```
if(b细分)
{
```

拥抱四叉树119

```
//否则,我们需要进一步向下递归到四叉树 if( !EdgeLength<=3 ) {
```

```
fChildOffset          = ( 浮动 )( ( iEdgeLength-1 )>> 2 );
iChildEdgeLength= ( iEdgeLength+1 )>> 1;

//细化各种子节点
//左下角
细化节点 (x-fChildOffset,z-fChildOffset,
           iChildEdgeLength,相机) ;

//右下
RefineNode(x+fChildOffset, z-fChildOffset,
           iChildEdgeLength,相机) ;

//左上
细化节点 (x-fChildOffset,z+fChildOffset,
           iChildEdgeLength,相机) ;

//右上方
细化节点 (x+fChildOffset,z+fChildOffset,
           iChildEdgeLength,相机) ;
}
```

这就是节点细化功能。基本上,您只是在检查细分并存储结果。如果需要细分,则进一步向下递归四叉树。我们列表中的下一个渲染

功能。

渲染四节点渲染是一个相当长的过程 (更不用说重复了),但不要让 demo6_1 (或本章的其他演示)中令人生畏的渲染函数大小欺骗了你;渲染比看起来要简单得多。

要启动该功能,我们将检查我们正在渲染的当前节点是否具有最高细节级别 (边长为 3 个单位或更少)。如果是,我们可以直接渲染它而无需特别注意。

120 6. 攀登四叉树

只要记得检查周围的节点是否是较低的LOD;如果是,请跳过风扇那一侧的顶点。

如果当前节点不是最详细的节点,我们还有更多工作要做。我们将计算渲染代码以确定为当前节点绘制什么样的三角形扇形。

为了计算这个渲染代码,我们将使用存储在四叉树矩阵中的传播值,如下所示:

```
//计算风扇的bit-iFanCode
//排列（需要渲染哪些粉丝） //右上角

iFanCode = (GetQuadMatrixData(iX+iChildOffset,
                               iZ+iChildOffset)!=0)*8;

//左上 iFanCode|= ( GetQuadMatrixData( iX-iChildOffset, iZ+iChildOffset)!=0 )*4;

//左下角
iFanCode|= (GetQuadMatrixData(iX-iChildOffset, iZ-iChildOffset)!=0)*2;

//右下 iFanCode|= ( GetQuadMatrixData( iX+iChildOffset, iZ-iChildOffset)!=0 );
```

在我们计算出渲染代码之后,我们就可以决定需要绘制哪些扇形了。我们将首先检查是否需要渲染风扇。如果要渲染当前节点的所有子节点,就会发生这种情况。另一个微不足道的情况是,如果只需要绘制左下/右上风扇或右下/左上风扇。最后一个简单的情况是,如果我们根本不需要渲染节点的任何子节点,我们可以将当前节点渲染为一个完整的三角形扇形。

除了我们刚才提到的琐碎案例之外,还有其他案例。

这些包括需要在四叉树上渲染的部分风扇。例如,在图 6.13 中,部分扇形需要渲染底部节点,该节点由两个禁用的子节点组成。

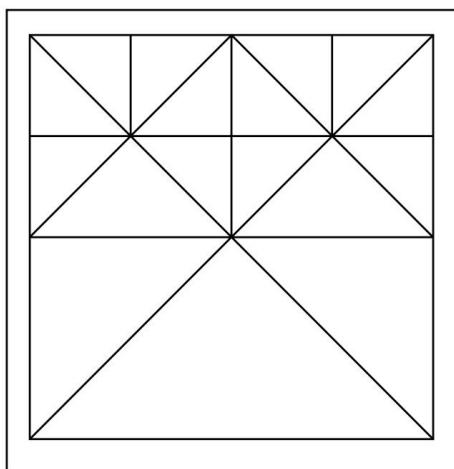


图 6.13 需要特别注意渲染的四叉树节点。

部分扇形渲染的代码是相当常规的,但它也是高度冗余和冗长的。如果您想查看它,请查看 demo6_1 中 quadtree.cpp 中的 RenderNode 函数。这一切都在那里,并且评论很多,所以它是不言自明的。

在绘制了部分扇形之后,我们需要弄清楚我们需要向下递归到当前节点的哪些子节点。完成后,我们就像金子一样!至此,基本的四叉树实现就结束了。去看看 demo6_1 和图 6.14 和 6.15。另请查看表 6.1 以了解演示的控件。

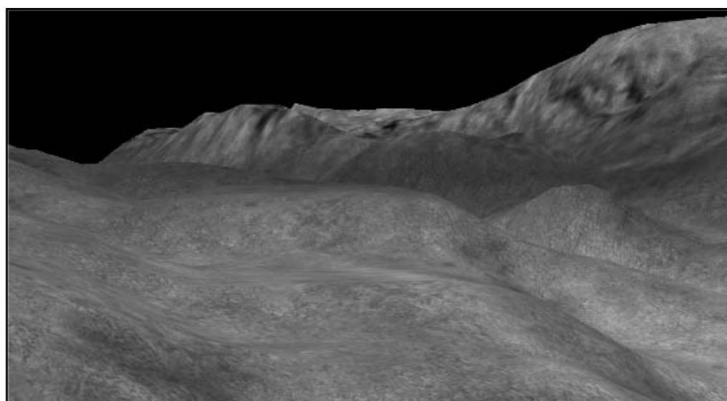


图 6.14 来自 demo6_1 的纹理和细节映射屏幕截图。

122 6. 攀登四叉树

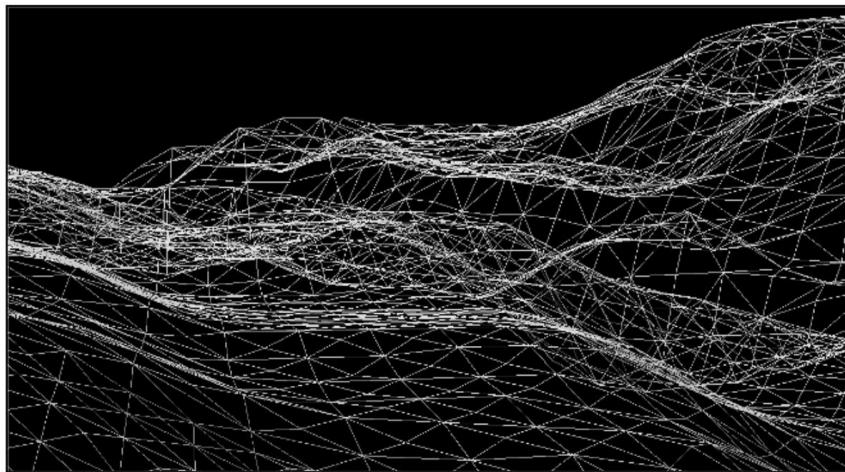


图 6.15 图 6.14 中地形的线框图像。

表 6.1 demo6_1 的控件

钥匙	功能
逃生/q	退出程序
向上箭头	前进
向下箭头	往后退
右箭头	向右扫射
左箭头	左扫射
吨	切换纹理映射
D	切换细节映射
在	以线框模式渲染
小号	以实体/填充模式渲染
+ / -	增加/减少鼠标灵敏度
] / [增加/减少移动灵敏度
1 / 2	增加/减少所需的分辨率
3 / 4	增加/减少最小分辨率

拥抱四叉树123

使事情复杂化现在是时候实现我们之前讨论的粗糙度传播了。为此，我们需要将四叉树矩阵的精度从布尔值矩阵增加到一系列字节（无符号字符）值。但是，除此之外，我们不需要更改第 5 章中的那么多代码。我们将添加一个您希望添加到初始化过程中的函数：
PropagateRoughness。

请记住,对于传播,我们希望采用自下而上的方法,这意味着我们希望从树的最高细节级别开始——边长为三个单位的节点。然后我们要遍历所有节点并传播它们的表面粗糙度。

```
//将 iEdgeLength 设置为 3 (可能的最小长度) iEdgeLength= 3;
```

//从最低级别的细节开始并遍历

```
//直到最高节点（最低细节） while( iEdgeLength<=m_iSize ) {
```

```
//节点边的偏移量（因为所有边的长度相同 iEdgeOffset= ( iEdgeLength-1 )>>1;
```

```
//节点子边的偏移量 iChildOffset= ( iEdgeLength-1 )>>2;
```

```
for(z=iEdgeOffset;z<m_iSize;z+=(iEdgeLength-1)){
```

```
for(x=iEdgeOffset;x<m_iSize;x+=(iEdgeLength-1)){
```

以此为循环的基础，我们可以计算当前节点的 d_2 值。正如我之前所说，您需要做的就是从两个角顶点获取当前近似值，然后将其减去该点的真实高度。以下是计算节点中上顶点的示例代码：

```
d2= ( int )ceil( abs( ( (
```

124 6. 攀登四叉树

计算不同的 d2 值后,您需要获取最大值并确保它在 0-255 的允许范围内。 (这是 unsigned char 可以获得的最大精度。我不了解你,但我不想花更多的内存来制作更高精度值的四叉树矩阵。)

我们想计算当前节点的一般表面粗糙度,而不是计算近似/实际高度。这很简单。

您要做的就是从当前节点的九个顶点中提取高度值。完成此操作后,您需要将计算的 d2 值存储在当前节点的四叉树矩阵条目中。

然后您可以使用最大高度和 d2 值进一步向上传播四叉树。

计算 f 时,新的四叉树值将应用于 RefineNode 函数,因此您实际上不需要更改太多代码。简单地编辑四叉树矩阵会使我们在第 5 章中使用的代码发生各种很酷的事情。现在去看看 demo6_2 控件与之前的演示相同,只是这一次对粗糙度较高的区域提供了更多细节,如图 6.16 中的线框图所示。

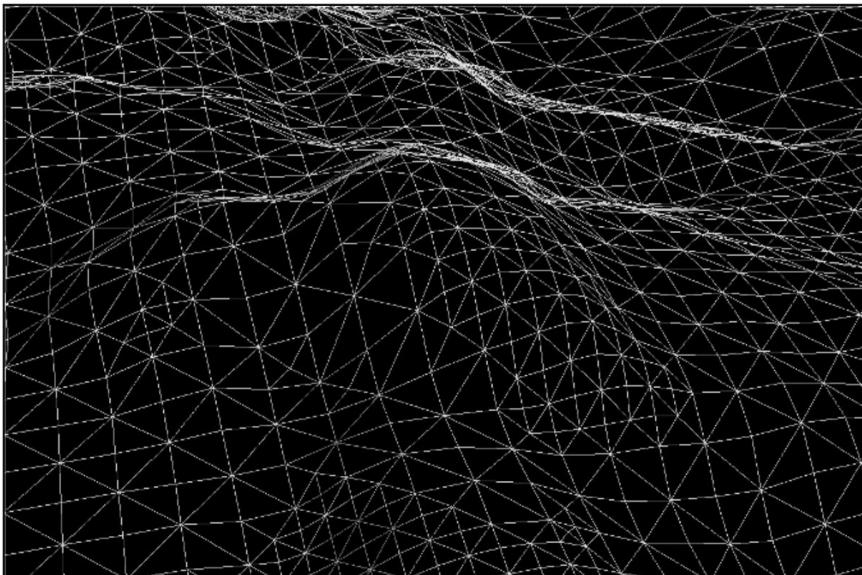


图 6.16 来自 demo6_2 的线框屏幕截图,显示了表面传播如何影响镶嵌网格。

总结125

加快速度 我们要做的最后一件事是在我们的实现中添加平截头体剔除,这是一种加快实现速度的简单方法。要添加平截头体剔除,我们需要编辑的唯一函数是RefineNode。我们的平截头体测试与第 5 章相同。我们将从当前节点制作一个“立方体”,然后针对视口对其进行测试。如果节点在视图中,我们将继续优化节点。如果它不在视图中,我们会将节点的四叉树矩阵设置为 0,并从更新和渲染队列中删除该节点及其所有子节点。

```
//根据视锥测试节点的边界框 if( !m_pCamera->CubeFrustumTest( x*m_vecScale[0],
GetScaledHeightAtPoint( x, z ),
z*m_vecScale[2],
iEdgeLength*m_vecScale[0] ) )
{
    //禁用这个节点,然后返回 (因为父节点被禁用,我们不需要浪费任何CPU //循环
    //向下遍历树) m_ucpQuadMtrix[GetMatrixIndex( ( int )x, ( int ) z )]= 0;
}

返回;
```

至此,我们结束了对四叉树算法和实现的介绍。查看 demo6_3 并见证您所有辛勤工作的成果。好工作!

概括

这是一个有趣的章节,我们几乎涵盖了与 Stefan Roetgger 的四叉树算法有关的所有内容。我们讨论了一般四叉树是什么,然后讨论了四叉树算法背后的所有理论。从那里,我们实现了所有的理论。我们最终得到了一个快速、灵活且美观的地形实现!我们只有一种地形算法要覆盖,所以让我们开始吧!

126 6. 爬四叉树

参考

1 Roettger, Stefan, Wolfgang Heidrich, Philipp Slusallek 和 Hans Peter Seidel。实时生成高度场的连续细节层次。在 V. Skala, 编辑, Proc。 WSCG 98, 第 315-322 页, 1998 年。<http://wwwvis.informatik.uni-stuttgart.de/~roettger/data/Papers/TERRAIN.PDF>。

第七章

无论在哪里

您可以

漫游

128 7.无论你在哪里漫游

在我们的 CLOD 地形算法覆盖的最后一部分中,我们是将介绍实时最优自适应网格(ROAM)算法。在过去的几年里,ROAM 一直是地形的代名词,但它最近受到了抨击,因为它被广泛认为对于现代硬件来说“太慢”。在本章结束时,你会感到震惊,任何人都认为它很慢!不过,现在让我们回顾一下本章的议程:

- ROAM 算法背后的理论 ■ Seamus McNally 的 ROAM 改进
- ROAM 新千年
- 实施新的和改进的 ROAM

议程现在可能看起来相当常规,但它绝不是。我们详细讨论了 ROAM (旧理论和新理论),并且我们在实施 ROAM 时也非常小心,以便我们能够获得最大的“物有所值”。我现在就闭嘴,这样我们就可以继续这一章了!

ROAM 算法

由 Mark Duchaineau 开发的 ROAM 算法 1在过去几年中一直是地形实施的标准。随着 Seamus McNally 的TreadMarks的发布,ROAM 的受欢迎程度飙升,它对经典算法的想法进行了一些新的转折,并让人们重新思考他们对 ROAM 的想法。所有这些以及更多内容都将在本章的第一部分进行讨论,所以让我们开始吧!

理论 ROAM 算法 (其
白皮书可以在 CD 上找到,Algorithm Whitepapers\roam.pdf)由一系列独特的
彻底改变地形可视化的想法。我们将介绍这些想法
在论文中介绍,从基本数据结构开始。

ROAM 算法129

二叉三角树ROAM 算法使用一种称为二叉三角树的独特结构来存储多边形信息。这棵树从一个粗根三角形开始（见图 7.1）。

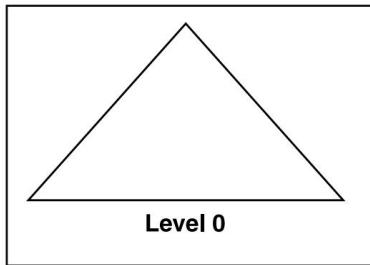


图 7.1二叉三角树节点的 0 级细分。

就像那个三角形看起来陈旧而粗糙,记住它是镶嵌的第一层 它不应该令人印象深刻。为了稍微向下遍历树,我们想要细分当前三角形 (0 级镶嵌)。为此,我们希望从三角形的三个顶点中的任何一个 “绘制”一条直线,该直线将与顶点相对的直线平分为两个相等的线段,从而形成两个底角为 90 度的三角形。这将产生由两个三角形组成的 1 级镶嵌 (见图 7.2)。

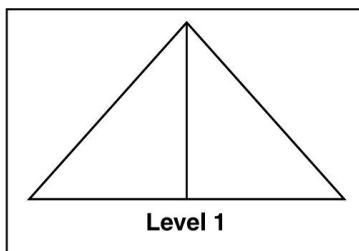


图 7.2二叉三角树节点的 1 级细分。

哇,惊人的两个三角形!我们需要进行另一个 “细分”通道（使用与之前细分相同的技术）。

130 7.无论你在哪里漫游

这样做会产生 2 级镶嵌,使我们的三角形计数达到 4。(如果您一直在感知一种模式,但不太确定每个细分的三角形增加,我会告诉您:数字每个细分的三角形加倍:1(图 7.1)、2(图 7.2)、4(图 7.3)、8(图 7.4)、16(图 7.5)等等。)

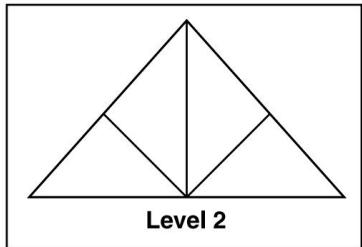


图 7.3二叉三角树节点的 2 级细
分。

这是另一个细分通道,它使我们的三角形总数达到 8 个。

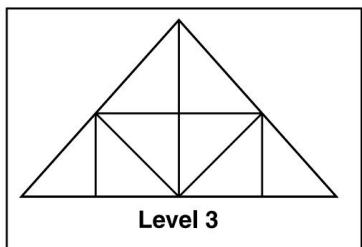


图 7.4二叉三角树节点的 3 级细
分。

在图 7.5 中,我们的三角形总数达到 16 个。细分不必止步于此;它们可以一直持续到达引擎底层高度图的分辨率为止。无论如何,前面的镶嵌只是为了展示单个二叉三角树节点的样本镶嵌的样子。但是,与您现在可能想的相反,实际的树节点不包含多边形信息。它只包含指向它的邻居和孩子的指针,稍后您会看到。

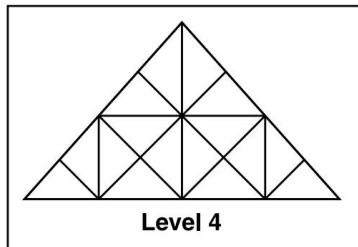


图 7.5二叉三角形树节点的 4 级细分。

镶嵌一系列二叉三角形树基础节点我们将用几个“基础节点”填充地形网格,这些基础节点将连接在一起形成一个连续的网格。像往常一样,开裂的怪物会在某个时候露出丑陋的脸。因此,在进行曲面细分时(从粗略级别到更详细级别,类似于我们在第 6 章“攀爬四叉树”中采用的自顶向下方法),我们可能必须“强制拆分”一个或两个节点。考虑图 7.6 所示的例子。

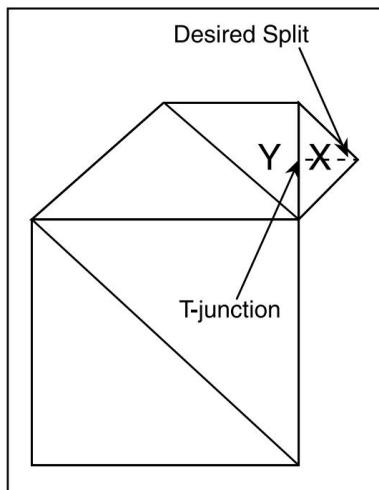


图 7.6即将发生的开裂问题。

132 7.无论你在哪里漫游

在图 7.6 中,我们想要细分三角形 X。但是,这样做,我们通过创建一个 T 形接头来造成裂缝,这是当一个三角形比相邻三角形具有更高的细节层次 (LOD),即如果我们要细分三角形 X 会发生什么。(T 形路口将由三角形 Y 形成。)为了防止这种结果,我们需要通过拆分图 7.6 中存在的其他三角形来强制拆分,直到它们具有统一的细节级别,并且不存在 T 形路口,因为如图 7.7 所示。

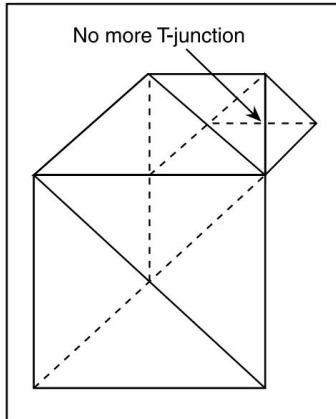


图 7.7 没有裂缝,没有 T 形接头,没问题!

拆分、合并和想象中的猫

好吧,我真的不确定想象中的猫是如何融入等式的
*把手套从桌子上扔掉.*把虚构的猫科动物移开,我们
可以处理下一个 也许是 最复杂的 部分
ROAM 白皮书:拆分和合并。 ROAM 论文建议
我们可以将当前帧网格基于前一帧的网格,而不是从头开始每一帧,并且
在需要的地方添加/减去细节。

为了完成这个任务,我们需要将三角树节点拆分成两个优先级队列:拆分队列和合并队列。这些队列将为镶嵌网格中的每个三角形保留优先级,从粗镶嵌开始,然后反复强制拆分或合并具有最高优先级的三角形。同样重要的是保持

要求子节点的优先级永远不会高于其父节点。

ROAM 算法的改进133

这是优先级队列的基本和基本解释,因为我现在不想花太多时间讨论它们。只要知道存在优先级队列并知道它们服务的基本目的。我们稍后会回到他们身边。

ROAM 算法的改进如果您还记得我们在第 1 章 “户外之旅” 中对 Seamus McNally 的TreadMarks的讨论,您会记得我说过 Seamus 如何通过在TreadMarks 中使用他的实现来真正提高 ROAM 的受欢迎程度。好吧,现在我们将深入了解他与传统 ROAM 算法究竟发生了什么变化的细节。 Bryan Turner 在 Gamasutra 上发表的一篇论文中也总结了这些想法。²您可以在 CD 上的 “算法白皮书”目录中找到该论文及其随附的演示。演示和论文都被压缩到 ROAM_turner.zip 文件中。

Seamus 的更改Seamus McNally 对 ROAM 算法进行了一些非常显着的更改,您可以在他的游戏TreadMarks 中看到这些更改。 Seamus 所做的更改通过减少 CPU 的工作负载和对二叉三角树节点使用相当酷的技巧来加快算法速度,同时也使算法对内存更加友好。以下是 Seamus 所做的一些更改:

- 没有为绘制的三角形存储数据 ■更简单的误差度量
- 没有帧到帧的连贯性

改进二叉三角形树节点Seamus 建议每个三角形节点不需要为每个渲染的三角形节点存储信息,而是需要很少的信息来完成其任务。该信息由与当前节点相关的三角形的五个“链接”组成(参见图 7.8)。

134 7.无论你在哪里漫游

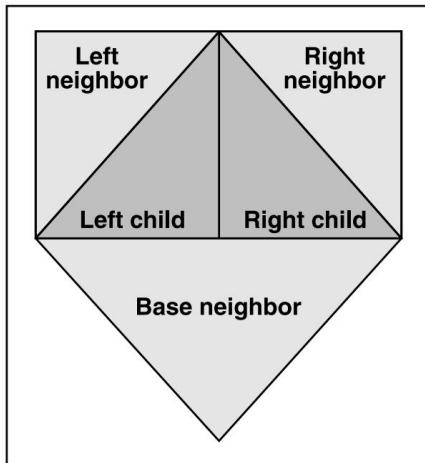


图 7.8 单个二叉三角树节点必须包含的信息。

如您所见,每个三角形只需要五个链接:两个链接到它的子节点(左右子节点)和三个链接到它的邻居节点(基、左和右邻居)。下面是一些简单的伪代码,如果你想实现它,结构会是什么样子:

```
结构 BinTriNode {
    BinTriNode* leftChild;
    BinTriNode* rightChild;
    BinTriNode* leftNeighbor;
    BinTriNode* rightNeighbor;
    BinTriNode* baseNeighbor;
}
```

要实际使用该结构,您可以在初始化时分配一个节点池,二叉三角形树可以在运行时从中绘制三角形。这几乎消除了地形的运行时内存分配,并控制了地形的详细程度。然后地形调用该节点池进行曲面细分和渲染,这

我们很快就会更详细地讨论。

简化误差度量

ROAM 白皮书中提出的错误度量由一系列复杂的数学例程组成(这就是为什么它不是

ROAM 算法的改进135

笔记

预先分配BinTriNode的内存池并不难。您只需声明一个指向BinTriNode缓冲区的指针（表示我们稍后将分配缓冲区），如下所示： BinTriNode* pTriPool;

然后，要分配整个缓冲区，请使用 C++ 的 new 运算符，如下所示：

```
pTriPool=new BinTriNode[numTris];
```

这里的所有都是它的！我只是想补充一下，以防您对此有疑问。

涵盖在我们的白皮书的简短封面年龄中）。然而，Seamus 提出了一种更简单的误差度量，可用于详细计算。（当您尝试决定是否拆分三节点以及应该拆分多深时使用错误度量。）

我们将使用的误差度量由一个简单的计算组成，并且完全“发生”在三角形的斜边上。

(如果您阅读第 5 章“CLOUD 受损的地理映射”和第 6 章“攀爬四叉树”中的变形部分，您应该对这个计算很熟悉。) 我们只是要计算平均长度的增量和真实长度。考虑图 7.9 中的三角形。

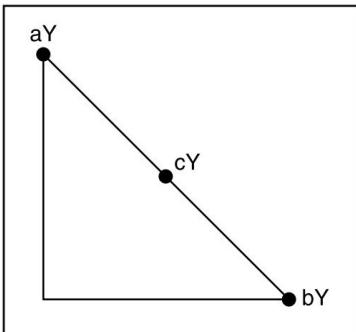


图 7.9 与误差度量计算说明一起使用的示例三角形。

对于计算，我们只需要标记顶点的高度分量。我们将计算 cY 处的近似值与实际值 cY 之间的差异，如图 7.10 所示。

136 7.无论你在哪里漫游

$$\text{metric} = \left| cY - \frac{aY + bY}{2} \right|$$

图 7.10 计算三角形误差度量值的公式。

就像我们在第 5 章中讨论的变形计算一样,我们真正在做的只是试图弄清楚有多少弹出

如果我们细分当前三角形,就会发生。我们可以确定

通过首先弄清楚变化有多大

如果三角形被细分,则会出现高度,然后投影

更改为屏幕像素。(后者需要一些相当复杂的

数学,真的没必要。真正需要做的是

计算世界空间的变化。)

仅拆分方法

几节前,我们讨论了中提出的拆分/合并想法

ROAM 白皮书。Seamus 提出帧到帧的连贯性

(使用拆分/合并从前一帧细分网格

优先队列)应该被完全消除。虽然加

拆分/合并支持(双优先级队列)可以增加灵活性

和你的应用程序的速度,这是一个高级主题,它有时会使程序员陷入困境。

如果您不基于镶嵌网格的网格,您会怎么做

从上一帧?好吧,你每个人都从头开始

框架并实现所谓的仅拆分曲面细分,它开始

在粗略级别 0 详细级别并细分到合适的级别

的细节。这种技术实际上比实现起来容易得多

听起来。

现在我将向您推荐 Bryan Turner 的演示和随附 CD 上的文章。Bryan 在一些易于阅读的代码中实现了 Seamus McNally 的许多改进。该 CD 还包括 Bryan 的

他的代码所基于的教程。您可以在

算法白皮书/ROAM_turner.zip 中的 CD。一探究竟!

漫游 2.0

是的,没错:ROAM 2.0。在本章中,我一直与 Mark Duchaineau (ROAM 算法的创建者)密切合作,以便文本和代码将向您介绍新算法的复杂性,在撰写本文时,该算法尚未发布。

我们将逐步进行这个解释,在每个步骤中混合一点理论和实现。在这些步骤结束时,我们将运行完整的 ROAM 实施。以下是我们将采取的步骤:

1. 实现基础
2. 添加平截头体剔除
3. 添加主干数据结构
4. 添加拆分/
合并优先级队列

第 1 步:实现基础这一步实际上就像标题所说的:基本。在本节中,我们不涉及任何复杂的内容;我们只介绍了我们将要编码的 ROAM 实现的基础知识,例如多边形镶嵌和其他有趣的东西。让我们开始吧!

像往常一样,我们将把实现分成四个高级函数,它们初始化、更新、渲染和关闭引擎。对于第一个实现,更新/关闭函数将几乎是可笑的小,每个由一行组成。为了与前几章保持一致,让我们从初始化函数开始,然后逐步介绍关闭函数。然而,与前几章不同,我将理论与实现结合起来。

初始化 v0.25.0

在初始化期间,只需要完成几个任务。首先,有必要快速描述前几个演示的一些细节。我们将使用我们在第 2 章 “Terrain 101” 中讨论的中点置换算法的一个版本,在程序上动态生成地形(没有高度图、没有光照、没有看起来很酷的纹理贴图,什么都没有)。这是中点位移的简化版本;它可以用一个数学方程来描述,如图 7.11 所示。

138 7.无论你在哪里漫游

$$\sum_{l=0}^{levelMax} md_l = \frac{scale}{\sqrt{2^l}}$$

图 7.11 计算每个细节级别的最大中点位移值的数学方程。

在等式中,l 是当前级别 (在循环中),levelMax 和 md 是我们存储当前级别 l 的缩放 (比例)计算的地方。对于那些以代码形式比数学形式更好地理解过程的人,这里是代码友好的版本:

```
for(lev=0;lev<=levelMax;lev++)
    MD[lev]= scale/( ( float )sqrt( ( float )( 1<<lev ) ) );
```

我们将在 ROAM 实施的前几个步骤中使用该表,因此您最好开始喜欢它。(CROAM 的表实例被命名为m_fpLevelMDSIZE以供将来的代码示例,仅供参考。)

本书中的下一段代码与实际的 ROAM 实现无关。它只是在程序上创建一个纹理,当应用于三角形时,会产生一个很酷的“线框式”

纹理网格。

更新 v0.25.0 好吧,我不太确定你能

处理这个巨大的函数,但我现在就告诉你,希望你看的时候不会因为函数的巨大尺寸而爆炸在 demo7_1 的代码中。在我的实现中,更新函数只包含……一个空的函数体!也许我们以后会有更多的更新。

渲染 v0.25.0

渲染函数被拆分成一个子渲染函数,用于递归渲染子节点 (类似于 RenderNode 函数

在第 6 章中), 所以问题变成了: 我们从哪里开始? 出色地,
我会让你轻松回答这个问题。 (相反, 我将提供
回答, 你可以假装你一直都知道。) 我们开始吧
具有高级渲染功能。

使成为

高级 Render 函数相当简单和常规, 它的
类似于第 6 章中的高级渲染函数。

在 CROAM::Render 中的工作是当我们试图找出根节点的顶点信息时, 它由
左下、右下、左上、
和右上角的顶点。完成后, 我们只需要渲染
使用递归 RenderChild 函数的两个基本三角形:

```
渲染子 (0, 绿色[0], 绿色[1], 绿色[3]) ;  
渲染子 (0, 绿色[3], 绿色[2], 绿色[0]) ;
```

您将了解函数参数的含义

其次, 但现在, 您只需要知道我们正在发送三角形的顶点信息, 以便该函数可以
渲染三角形或进一步向下递归并为下一个三角形重复该过程。

渲染子

这个函数比前面的函数稍微多一些

到目前为止, 我们已经讨论过, 但是当我们到达这个实现的最后一步时, 它并
没有那么复杂。

该函数需要接受四个参数, 其中三个是顶点
信息, 其中第四个是当前级别
呈现。 (我们需要关卡信息, 以便我们可以深入了解
我们之前创建的中点位移表。)

函数, 我们可以存储最大中点位移值
局部变量 fMD 中的当前级别, 以便事情变得更容易。然后
我们可以计算分割父三角形时形成的新顶点 (见图 7.12)。

完成后, 我们可以进行一些棘手的 IEEE 浮点计算。简而言之, IEEE 浮点运
算用于最小化存储并最大化某些变量和某些变量的速度

计算。在我们的例子中, 我们想要计算一个随机扰动
在新顶点中, 该顶点由当前三角形的父级分割高度值 (Y 分量) 形成。为
此, 我们将

140 7.无论你在哪里漫游

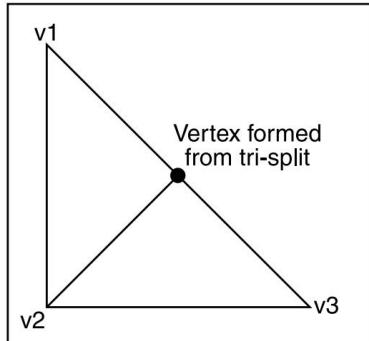


图 7.12 由当前三角形的父级分割形成的顶点。

从填充了随机整数变量的哈希表中提取一些值，并用该信息填充本地无符号短变量：

```
pC= ( unsigned char* )fNewVert;
for(i=0, uiS=0;i<8;i++)
    uiS+= randtab[( i<8 )
                    | pC[i]];
```

randtab是随机哈希表， pC是 unsigned char 版本新顶点的信息。

在我们用我们的值填充本地无符号短变量uiS 之后

哈希表，我们需要将这些值转换为浮点值。

使用 IEEE 浮点技巧将整数值类型转换为浮点值比典型的类型转换快得多：

```
fFloat = ( 浮动 )iInt;
```

但是，使用 IEEE 浮点技巧对整数变量进行类型转换

到一个浮点变量，虽然更快，是一个很大的丑陋和

对于不知道发生了什么的人来说，它更加神秘。

所以，对于那些知道发生了什么的人来说，这很棒。但是，如果它让您感到困惑，请查看侧边栏。（而且，要更严格地介绍 IEEE 浮点运算，请查看 Game 中的 gem 2.1

Programming Gems 2，这是对该主题的一个很好的介绍。）

```
pInt= ( int* )( &fRandHash);
```

```
*pInt= 0x40000000+( uiS & 0x007fffff );
```

```
fRandHash-= 3.0f;
```

笔记

这个先前的 IEEE 浮点代码示例获取浮点 fRandHash 的地址并将其分配给指针 pInt。如果 pInt 是一个 32 位整数,其中 31 是高位,0 是低位,那么我们

通过分配 0x40000000 来设置第 30 位,它的指数为 128。然后我们在 uiS 上添加“与运算”的结果,然后
0x007FFFFF。

完成“和操作”以屏蔽任何可能侵入的位

在指数位上。在 IEEE 浮点标准中,第 31 位是符号:0 表示正数,1 表示负数。位 30-22 是

指数,位 21-0 是小数点后的十进制值

观点。因此,如果 uiS 包含 0x000F0000 (十进制为 983,040),我们的浮点数
fRandHash 将等于 2.23438,然后我们
减去 3 得到 ×1.23438,整数格式为
0x40400000。

基本上,所有这些都是为了加快冗长的操作,例如
类型转换,通过使用一些低级位移。这就是全部
就是它!我希望这个小教程能帮助你理解到底发生了什么。

我们想将之前的计算应用到新顶点的高度
价值。为此,我们将使用图 7.13 中的等式。

$$\text{newY} = \frac{y1 + y3}{2} + \text{randHash} * MD$$

图 7.13 计算新顶点高度值的公式。

像往常一样,下面是与图 7.13 中相同的等式,除了
代码形式:

```
fNewVert[1] = ((fpVert1[1]+fpVert3[1])/2.0f)+fRandHash*fMD;
```

我们需要计算从相机的眼睛位置到
新顶点的位置。执行正常距离的所有步骤

142 7.无论你在哪里漫游

除平方根步骤外的公式。保持值平方。这通过避免调用 `sqrt` 为我们节省了一些宝贵的时间。计算出距离后，我们需要决定是否要对当前节点进行细分。我们必须满足两个要求来细分节点。首先，我们需要确定细分到另一个节点是否需要超过最大细节级别。如果是这样，那么我们不能细分；我们只需要渲染当前的三角形。其次，我们需要查看观察者是否足够近以打扰细分。

如果满足这两个要求，我们可以递归到当前三角形的两个孩子：

```
//看能不能细分当前节点
if(iLevel<m_iMaxLevel && SQR(fMD)>fDistance*0.00002f)
{
    //渲染孩子
    RenderChild(iLevel+1, fpVert1, fNewVert, fpVert2);
    RenderChild(iLevel+1, fpVert2, fNewVert, fpVert3);

    //当前节点不需要渲染
    //因为它的两个孩子都是
    返回;
}
```

大多数顶点名称应该是不言自明的。正如我之前所说，如果不满足细分的要求，我们可以使用作为函数参数传递的三个顶点来渲染当前三角形。这就是功能！

关机 v0.25.0

在关闭例程中，我们需要做的就是释放我们为中点位移表分配的内存。这就是关闭例程所需的全部内容。希望你没有眨眼！

演示 v0.25.0

好吧，我们已经到了第 1 步的结尾。最终的结果很酷，虽然有点简单。不过，我们将在实现的最后几个步骤中弥补这种简单性。现在，去看看 `demo7_1`。控件在表 7.1 中列出，`demo7_1` 的屏幕截图（在 CD 上的 `Code\Chapter 7\demo7_1` 下）如图 7.14 所示。

表 7.1 演示 7_1 的控件

钥匙	功能
逃生/q	退出程序
向上箭头	前进
向下箭头	往后退
右箭头	向右扫射
左箭头	左扫射
+	增加最大网格级别
×	降低最大网格级别

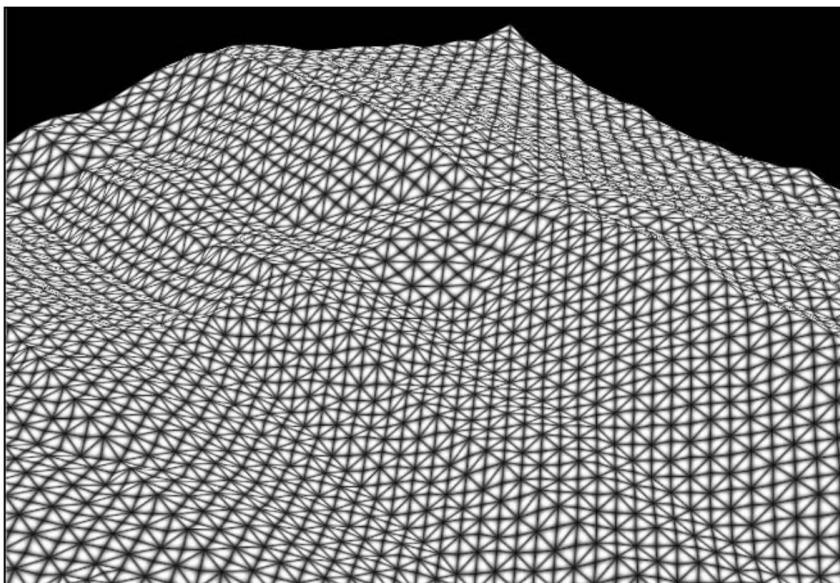


图 7.14 来自 demo7_1 的屏幕截图,ROAM 实现的第一步,
它显示了算法的基础知识。

是时候提醒你一些事情了。我在本书附带的 CD 上提供给你的实现只是
为了给你一个

144 7.无论你在哪里漫游

关于如何实施上述技术的想法。它们绝不是完美的、高度优化的,甚至不是尽可能详细的。

因为地形这个话题远非静态的,你可以实现某种算法的各种方式会随着时间的推移而发生很大的变化。要获得最高度优化、最佳外观和简单酷炫的算法实现,请查看我的网站 <http://trent.codershq.com/>,在那里您应该可以找到您正在寻找的东西。话虽如此,让我们继续下一部分!

第 2 步 :添加平截头体剔除 在这一步中,我们将实现平截头体剔除,这与我们在前面章节中所做的方式略有不同。与上一个演示不同,我们还将使地形脱离高度图(并使用纹理贴图)。让我们开始吧!

初始化 v0.50.0

初始化几乎与第 6 章中的相同,只是我们将摆脱程序网格纹理生成。相反,我们只是要生成中点位移大小表,然后继续。

渲染 v0.50.0

渲染功能发生了一些变化。我们不会在 $[-1, 1]$ 范围内为基础三角形生成顶点,而是在 $[0, \text{size}-1]$ 范围内生成它们,其中 size 是地形底层高度图的大小。这是使引擎脱离高度图所需的主要步骤。完成后,我们将像上次一样进入 RenderChild 例程。

RenderChild,重温好的,这个函数有一些

变化,将大大提高我们引擎的整体性能和渲染网格的整体“外观”。首先,让我们介绍一下性能升级。

像你从未有过的那样剔除……哦,没关系

当我们向这个引擎添加截锥体剔除时,我们所做的事情与前两种方法略有不同。我们将使用一个

边界球来描述我们三角形的大小。然后我们将针对视锥体测试该球体的包含/排除。我们需要知道边界球体的两件事：它的中心和它的半径（见图 7.15）。

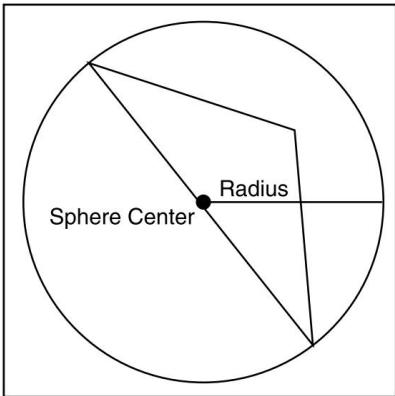


图 7.15 包围单个三角形的边界球。

要将这个想法应用到我们的代码中，我们需要当前三角形的顶点信息。我们需要知道构成三角形的三个顶点。然后我们需要计算斜边的中心（边界球的中心）。然后我们将找出离中心最远的顶点 这就是我们的半径。以下是此类计算的代码：

```
fSqrBoundTemp= SQR( ( fpVert2[0]-fNewVert[0] ) )+
    SQR( ( fpVert2[1]-fNewVert[1] ) ) +
    SQR((fpVert2[2]-fNewVert[2]));
```

```
//检查这是否是最大距离
//到目前为止我们已经计算过了
如果 (fSqrBoundTemp>fSqrBound)
    fSqrBound= fSqrBoundTemp;
```

在我们知道之后，我们可以轻松地测试球体是否包含在视锥中。如果包含球体，我们可以继续渲染函数。如果不是，我们可以立即返回；如果父三角形不可见，则其子三角形、子三角形的子三角形或子三角形的子三角形都不可见。这是代码

146 7.无论你在哪里漫游

我们将在本章剩余的演示中使用截锥体剔除。

如您所见,它与我们一直使用的技术有些不同,但速度也快了很多。在这段代码中,我们将针对每个平截头体平面测试一个点。第二个我们发现该点不在截锥体内,我们可以退出测试。这种技术大大加快了我们的应用程序。

```

如果 (iCull!=CULL_ALLIN)
{
    浮动 r;
    整数 j,米;

    //对视锥执行剔除 for( j=0, m=1; j<6; j++, m<<= 1 ) {

        if( !( iCull & m ) )
        {
            r= m_pCamera->m_viewFrustum[j][0]*fNewVert[0] + m_pCamera-
                >m_viewFrustum[j][1]*fNewVert[1] + m_pCamera->m_viewFrustum[j]
                [2]*fNewVert[2] + m_pCamera->m_viewFrustum[j][3];

            //检查片段是否包含
            如果 (SQR (r)>fSqrBound){

                //检查三角形是否真的在//在视锥体内

                如果 (r<0.0f)
                    返回;

                //三角形在视图内 iCull|= m;

            }
        }
    }
}

```

正如您在前几章中所了解的,平截头体剔除可以加快任何应用程序的速度,而且这里的情况并没有改变。通常,添加 frustum culling 可以提高任何应用程序的速度……除非你有一个非常臃肿的剔除程序。

纹理映射、细节映射和高度图

将先前命名的组件添加到引擎不需要很多工作,因为我们将网格的坐标转换为

高度图的。我们只需要像以前一样渲染一切
在过去的几章中。在高级渲染中绑定纹理
函数并计算当前顶点的阴影/高度值。

所有这些都在 demo7_2 中,所以如果你对纹理映射有点模糊
前几章的技巧,现在就去看看吧。在
事实上,查看 demo7_2 (在 CD 上的 Code\Chapter 7\demo7_2 下) ,
表 7.2,以及图 7.16 和 7.17。

表 7.2 演示 7_2 的控件

钥匙	功能
逃生/q	退出程序
向上箭头	前进
向下箭头	往后退
右箭头	向右扫射
左箭头	左扫射
+	增加最大网格级别
×	降低最大网格级别
吨	切换纹理映射
D	切换细节映射
在	以线框模式渲染
小号	以实心/填充模式渲染

148 7.无论你在哪里漫游

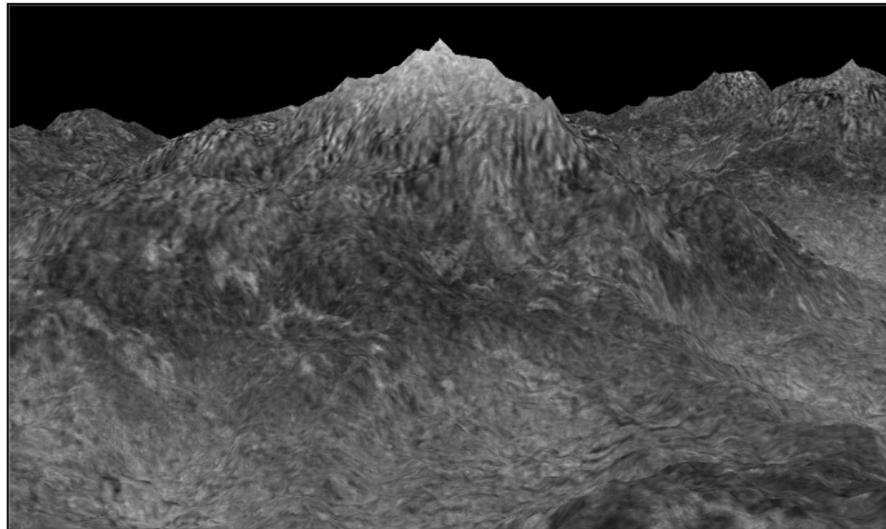


图 7.16 来自 demo7_2 的纹理映射和细节映射屏幕截图。

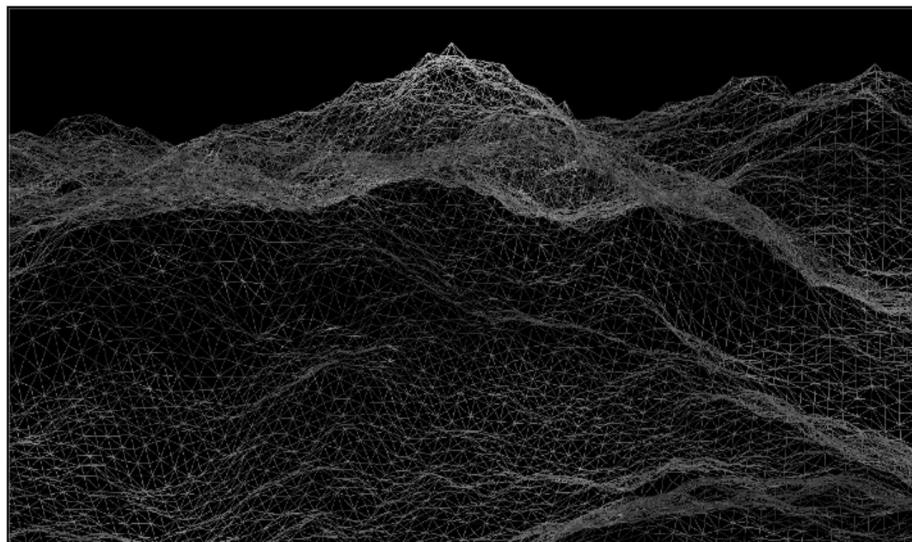


图 7.17 图 7.16 的线框版本。

第 3 步 :添加 骨干数据结构

过去的几个步骤是对 ROAM 2.0 基本概念的简单介绍。现在我们将创建未来步骤的核心骨干。与“老式”的 ROAM 算法不同,ROAM 2.0 依赖于菱形树骨干。虽然在概念上,这类似于我们在本章前面讨论的二叉三角树,但实现它却大不相同。让我们试试吧!

钻石是程序员最好的朋友 ROAM
2.0 实现的基本“单元”称为钻石。

树中的每个菱形都由两个连接在公共底边上的直角等腰三角形组成。每个三角形还包含四个子三角形 但我们有点超前了。让我们只分析图 7.18 中的基础菱形。

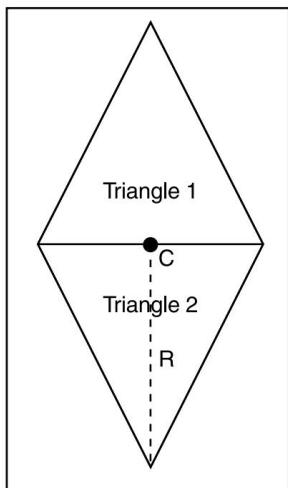


图 7.18 ROAM 实现中使用的基
本菱形的简单图像。

看?没什么特别的。我们有一个由两个三角形(三角形 1 和三角形 2)组成的简单菱形。钻石的中心顶点(图 7.18 中的 C)标识了每颗钻石。正如我们在上一节中讨论的那样,每颗钻石还包含其平方边界球半径,

150 7.无论你在哪里漫游

一个错误度量,它的分辨率级别（它在菱形树中的位置）,以及它的平截头体剔除位标志（在第 4 步之前我们不需要使用它）。每个菱形还包含一系列链接,如图 7.19 所示。

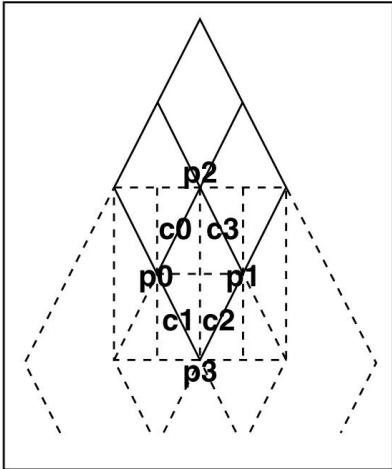


图 7.19 菱形及其父/子链接。

如图 7.19 所示,三角形网络从与图 7.18 相同的基础菱形开始,除了我们在它下面显示了两个菱形（用虚线表示）。首先,让我们从儿童开始。子项 (c_0, c_1, c_2 和 c_3)都是图 7.18 中原始基础菱形的子项。然后我们更详细地分析子 c_1 ,显示其父链接 (p_0, p_1, p_2 和 p_3)。父母 p_0 和 p_1 是孩子的左/右父母,父母 p_2 和 p_3 是孩子的上/下“祖先”父母。随着您对整个 ROAM 2.0 “系统”的熟悉,整个钻石的概念变得更加明显。你只能通过深入研究我们将要处理的实现来做到这一点。

基础钻石结构相当常规。您已经知道组成它的所有组件,因此接下来呈现的菱形结构伪代码应该不会太令人惊讶:

```
结构 ROAMDiamond {
    ROAMDiamond* pParent[4], pChild[4]
    ROAMDiamond* pPrevPDmndnd, *pNextPDmndnd
```

```

浮动中心[3];
浮动绑定;
浮动错误;
短 PQIndex;
int8 childIndex[2];
int8 级别;

uint8 剔除;
uint8 标志;
uint8 锁定计数;
int8 填充[4];
};

这很简单,不是吗?嗯,这是我们 ROAM 2.0 实施的第 3 步和第 4 步的基础,
所以你最好习惯这种结构!

```

创造钻石啊,要是我能创造自己的钻石

就好了……谈论赚钱的人!无论如何,我们将讨论一些可以“创建”新钻石孩子所需信息的伪代码。我们要讨论的这个函数是第3步的基础,所以你最好注意!

在菱形子创建函数中,我们唯一最重要的目标是该函数为菱形子生成链接以保持网格一致。尽管第 3 步不提供本地裂缝固定支持,但将网格的菱形链接在一起仍然很重要。(将孩子与父母联系起来,反之亦然。)这是我们创建函数的主要目标。当然,我们也想初始化孩子的信息。毕竟,如果孩子什么都不知道,那还有什么意义呢?

```
ROADMond CreateChild( child, index ) { // 如果已经存在则返
```

```
回 if (child->pChild[index]) return child->pChild[index];
```

```
//分配新的
```

```
k=分配钻石 () ;
```

152 7.无论你在哪里漫游

```
// 递归地为孩子 i 创建另一个父级 if (index<2) {

    px=子->pParent[0]; ix= (child-
    >childIndex[0]+( index ==0 ? 1 : -1 )) & 3;
} else { px=
    child->pParent[1]; ix= (child-
    >childIndex[1]+( index ==2 ? 1 : -1 )) & 3;
}
cx= CreateChild( 像素, ix );

//设置孩子的链接
孩子->pChild[i]= k; ix= ( 我 &
1 )&1; if (cx->pParent[1] == px)
ix|= 2; cx->pChild[ix]=k;

如果 (索引&1){
    k->pParent[0] = cx; k-
    >childIndex[0]= ix;
    k->pParent[1] = 孩子; k-
    >childIndex[1]=索引;
} 别的 {
    k->pParent[0]=孩子; k-
    >childIndex[0]=索引; k->pParent[1]
    = cx; k->childIndex[1]= ix;
}

k->pParent[2]= child->pParent[index>>1]; k->pParent[3]=
child->pParent[(( ( 索引 + 1 ) & 2 )>>1 ) + 2];
ResetChildLinks();

// 计算孩子等级,顶点位置 k->level = child->level + 1; k-
>center=中点 (k->pParent[2]->center,
    k->pParent[3]->center);

计算边界半径 () ;
```

```
UpdateDmndCullFlags();  
  
    返回 k;  
}  
  
呸!这是很多伪代码和很多丑陋的小点  
转移/掩蔽操作!好吧,永远不要害怕。这一切都比看起来简单得多。  
所有的位移和掩码都用于确定孩子相对于其父母的方向。我们可以稍微清理一下这些丑  
陋的东西,  
但是通过位移而不是除法/乘法,我们加快了速度  
位(不是很多,但足以在常用的  
功能)。另外,所有这些小操作应该让你感觉很酷。
```

一起塑造主干钻石树

好的,您知道将步骤 3 放在一起所需的主要内容,
但你所掌握的知识有些零散,需要
“放在一起。”这就是本节的目标,所以让我们开始吧!

钻石池

菱形池是菱形结构的动态分配缓冲区。这个池是你在运行时“调用”的,当你

网格需要一个新的钻石。分配此池后,您
需要几个函数来管理您想要的钻石
利用。例如,如果您想创建一个新的钻石,您需要
从池中得到它。当你使用那颗钻石时,你不会
想在代码的其他地方使用相同的菱形。有必要创建几个“安全”功能:一个功能锁定一
个
使用钻石和另一个解锁钻石使用的功能。

锁定功能的工作只是移除未锁定的钻石
来自钻石的“免费清单”(钻石池)。为此,我们
需要找到最近解锁的免费钻石(应该
作为锁定函数的参数提供),将其作为我们的
使用,然后将旧的“最近”解锁的钻石重新链接到
下次我们要锁定钻石时使用不同的钻石
利用。解锁函数使用类似的方法,除了,你做的事情与锁定函数相反。

我们可以使用更多的功能来让我们的生活更轻松,那就是
将是一个钻石创建函数,它创建一个级别

154 7.无论你在哪里漫游

钻石池上空的抽象。创建函数只需要获取一个指向最近释放的钻石的指针。如果没有钻石可以“抢到”，那我们手上就有小问题了……

然而，大多数时候，我们不必考虑这一点，所以不要太担心。然后我们想知道这颗钻石以前是否被使用过。为此，我们可以使用菱形结构的成员变量之一作为“以前使用过”标志。

为此，我们将使用边界半径变量。在初始化时，我们将此变量设置为 $\times 1$ ，如果使用菱形，它将沿线的某处设置为不同的值。（这个值肯定会大于 0，当然，除非您看到一个具有负半径的球体，从而将其拉伸到任何 3D 坐标系的巨大未知数中。）无论如何，如果我们将钻石 `re_grabbing` 之前已经使用过，我们需要重置它的主要父/子链接，并确保从池中解锁它的父钻石。然后我们可以继续锁定抓取的钻石并将锁定的指针返回到我们可以玩弄的新创建的钻石。

有了这些池操作功能，我们在 ROAM 2.0 实现的菱形池主干上有了一个很好的抽象层。现在我们可以在步骤 3 中开始编写一个工作实现，而不用担心所有这些理论和伪代码。万岁！

初始化 v0.75.0

第 3 步的初始化函数比第 2 步要复杂得多。

（当然，第 2 步的初始化函数比第 1 步中介绍的要简单得多，所以现在您要为初始化中的“幸运休息”付出代价。）我们还有更多的“维护”工作要做，才能启动和运行演示。我们必须初始化钻石池，处理两个级别的基础钻石（更不用说将它们连接在一起），还有一大堆其他有趣的东西会让你大吃一惊。嗯……好吧，也许它不会让你大吃一惊。事实上，我想我什至会尝试让整个事情变得容易学习。我们走吧！

首先，我们需要初始化钻石池的内存。这并不难，我认为你可以自己处理。完成之后，我们需要进行一些“池清洁”，这可能会变得棘手。首先，我们要遍历所有池中的钻石，并将上一个/下一个链接设置为与当前钻石相关的相应钻石。请参见图 7.20。

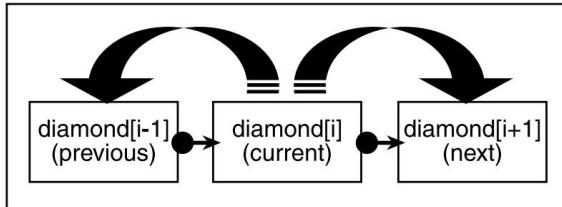


图 7.20 通过将每个节点链接到前一个/下一个节点来设置菱形池。

建立链接后,我们可以重新循环通过池并
初始化每颗钻石的“关键”变量。关键变量和
需要做的事情如下:

1. 包围半径必须设置为 $\times 1.0$, 这标志着
钻石节点为新的。 (实际上你可以使用任何其他小于 0 的浮点
值。如果你觉得你甚至可以使用 $\times 71650.034$
需求。)
2. 钻石的锁计数必须设置为 0, 也标志着
节点为新的和未使用的。

接下来,我们必须初始化网格的基础菱形。我们有两个
要初始化的钻石级别:一个 3×3 级别 0 钻石底座和一个 4×4
1 级钻石底座。两者都需要稍微不同的计算来确定钻石的中心, 并且每个都需要不同
的链接
技术, 但除此之外, 它们基本上需要相同的设置
程序。钻石的中心顶点将在范围内初始化
[$\times 3, 3$], 因此根据高度图的大小缩放这些值很重要。我们还需要计算钻石的等级,

这并不像看起来那么简单。很少涉及基础钻石
在 mesh 的实际渲染过程中, 它们实际上是取负值的。基础钻石只是用作“起点”

其余的网格。尝试渲染基础钻石将导致
在不幸的错误中, 这绝不是一件好事。我们拍完之后
照顾基础菱形初始化的第一部分, 我们需要设置
基础钻石链接, 但所有这些都是相当常规的。

渲染 v0.75.0

子渲染功能与上一步几乎相同, 但不是为每个三角形发送顶点信息

156 7.无论你在哪里漫游

单独地,我们将发送钻石信息并使用钻石中包含的顶点(钻石的中心顶点及其上一个和下一个钻石链接的中心顶点)。高级渲染功能变得更加简单。我们不计算基础三角形的顶点,而是简单地使用我们在初始化函数中初始化的基础三角形的信息:

```
//渲染网格
RenderChild(m_pLevel1Dmnd[1][2], 0);
RenderChild(m_pLevel1Dmnd[2][1], 2);
```

这就是渲染网格的全部内容。我们只需从1级基础钻石组中取出中间的两颗钻石并渲染它们的基础子级。这里的所有都是它的!去看看 demo7_3 (在 CD 上的 Code\Chapter 7\demo7_3 下)。您不会看到与 demo7_2 有太大的视觉差异 (如图 7.21 所示),因为我们所做的只是更改引擎运行的“背景”数据结构。您甚至不会注意到程序的速度有太大变化。

这一步主要是设置接下来两步将要运行的钻石树主干。无论如何,享受演示!

第 4 步:添加拆分/合并优先级队列这是
我们的实现现在速度和基础架构方面获得巨大升级的地方。不是在
每一帧之后重新细分网格,我们将在程序开始时进行主要细分,然后通过在需要
的地方拆分/合并菱形,将新细分的网格与前一帧的网格分开。在阅读本节之前,您
必须了解我们在上一节中讨论的金刚石骨架结构,因为本节广泛使用该结构。

优先级队列的要点你可能还记得本章前面的这个话题,除了那时
我们谈论的是三角二叉树而不是菱形;但是,我们讨论的基本概念是相同的。优
先级队列为拆分/合并钻石提供了一个“桶”。桶上最上面的钻石是优先级最高
的钻石,所以它会得到第一个拆分/合并处理。使用这些优先级

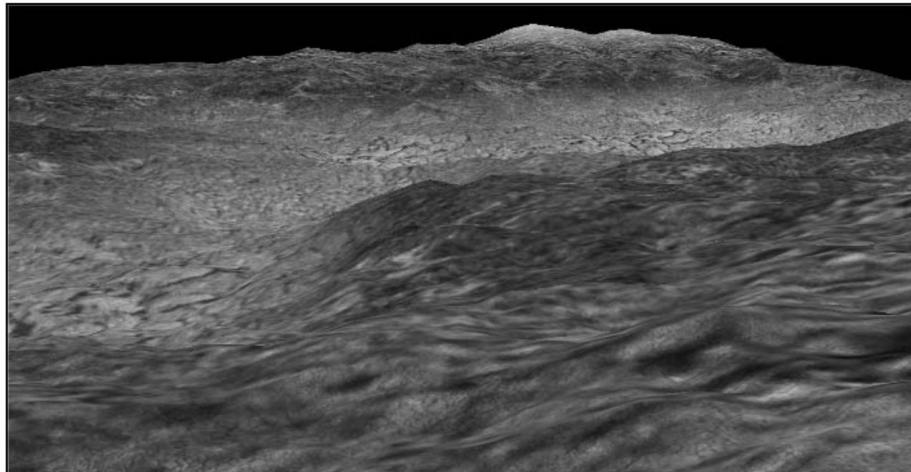


图 7.21 来自 demo7_3 的屏幕截图,我们在 ROAM 实现中添加了菱形主干。

队列,我们不会被迫为每一帧重置和重新镶嵌一个新的网格;因此,我们可以保持更刚性的多边形结构、更一致的帧速率以及各种其他优点。

我们将为第 4 步实现一个双优先级队列:一个合并队列和一个拆分队列。拆分钻石会导致更高级别的细节,而合并钻石会导致更低级别的细节。

通过将必要的拆分/合并拆分为两个单独的队列,我们无需在单个存储桶中对一大堆拆分/合并优先级进行排序,从而加快了流程。既然我们知道拆分/合并优先级队列结构的要点,那么我们究竟如何去实现它呢?好吧,现在是讨论这个问题的好时机!

实现拆分/合并优先级队列为了开始

我们的拆分/合并队列实现,我们首先需要创建两个菱形

指针数组 一个数组用于拆分队列,一个数组用于合并队列。队列保存菱形指针信息而不是实际的菱形数据。引擎将使用此菱形指针访问菱形的信息以对其进行拆分或合并。我们将给每颗钻石一个索引到拆分或合并数组中,以使我们的生活更轻松一些。

158 7.无论你在哪里漫游

首先,我们需要两个函数来更新钻石的优先级或钻石的队列索引。我们将讨论“优先级更新”函数,该函数采用菱形并根据当前视点的信息更新其优先级队列索引。

优先级更新函数采用钻石指针并根据与视点相关的信息(主要是从钻石中心到视点的距离以及与钻石距离相关的误差度量)更新其索引。我们希望通过检查结构中某处的标志来确保该过程尚未在菱形上完成。然后,考虑到该过程尚未对给定的钻石执行,我们继续进行距离/误差计算。钻石的误差值应该在创建时就已经计算好了,这样我们的生活就更轻松了。但是,我们需要根据与钻石与相机的距离相关的投影误差值来计算钻石的优先级。之后,我们需要调用下一个函数来用钻石的新索引更新优先级队列,并用新的索引替换队列中钻石的旧索引。这样做会导致我们讨论我之前谈到的第二个功能。

第二个函数将被称为“入队”,我们通过用新条目替换队列中的旧条目来更新菱形在其优先级队列(拆分队列或合并队列)中的条目。(新条目在优先级队列中的位置被定义为Enqueue函数的参数。)至于菱形在哪个队列中,该信息由菱形结构中的一个标志提供,这使得过程更加容易!

对于这个函数的第一部分,我们只关心从队列中的旧位置移除钻石。完成后,所有必要的队列标志和链接都已解决,我们希望将菱形插入优先级队列中的新位置,并使用新的队列信息更新菱形的标志。(我们实际上可能将钻石从一个队列移动到另一个队列,因此我们可能会将先前在拆分队列中的钻石移动到合并队列,反之亦然。)就是这样!这两个函数是管理优先级队列的主要菱形操作函数。

问题是我们在使用拆分/合并队列中存在的菱形时缺少两个重要函数:拆分函数和合并函数。

我们将首先讨论split函数,因为它非常简单。此函数将菱形指针作为参数（菱形指针是指向要拆分的菱形的指针）,如果尚未拆分,则将其拆分。为此,请查看以下伪代码:

```
拆分 ( ROADMond* dmnd ) {  
    //递归拆分父节点  
    for(int i=0;i<2;i++){  
        p= pDmnd->pParent[i];  
        拆分 (p) ;  
  
        //如果 pDmnd 是它的第一个孩子,则从合并队列中取出 p if( !( p->splitflags & SPLIT_K ) )  
  
        入队 (p,ROAM_UNQ,p->queueIndex) ;  
  
        p->splitflags|= SPLIT_K0<<pDmnd->i[i];  
    }  
  
    //获取孩子,更新剔除/优先级,并放入拆分队列  
    for(i=0;i<4;i++){  
        k=GetChild(pDmnd, i);  
        更新剔除 (k) ;  
        更新优先级 (k) ;  
  
        //新分裂的钻石的孩子进入分裂队列 Enqueue( k, SPLITQ, k->iq ); s= ( k->p[1]==pDmnd ?  
        1 : 0 ); k->splitflags|= SPLIT_P0<<s;解锁 (k) ;更新Tris(k);  
  
    }  
}  
  
//指示钻石被分割,更新排队,添加到清单 pDmnd->flags|= SPLIT; //新拆分的pDmnd继续  
mergeq  
  
入队 (pDmnd,MERGEQ,pDmnd->iq) ;  
}
```

合并函数是反向的拆分函数。如果你理解了split函数,那么merge函数应该就没有问题了!

160 7.无论你在哪里漫游

这处理了拆分/合并优先级队列系统的大部分细节！

三角形系统在这一步中,我们正在围绕渲染

系统进行相当多的更改。

因为我们并没有在每一帧从头开始完全重新网格化网格,所以我们真的不需要每帧更新三角形信息(即将发送到API的三角形)。我们将实现一个“三角形树”来跟踪要渲染的列表中的三角形。此三角形树由一个大型浮点数组定义,该数组用作一种顶点缓冲区,用于将三角形发送到渲染API。我们也会在这个数组中存储纹理坐标。但是,为了让我们更轻松并清理代码,我们需要提出操作函数来管理信息。我们将需要一个函数来处理以下每个任务:

- “分配”一个新三角形并将其添加到列表中 ■ 从列表中
- “释放”一个分配的三角形 ■ 将一个新三角形添加到列表
- 中 ■ 从列表中删除一个三角形

记住:我们实际上并没有在这个系统中分配/释放内存 看起来我们是!分配/释放函数是调用添加/删除三角形函数的高级抽象。让我们专注于低级操作函数,因为高级操作很容易解释。

添加/删除功能相互补充,因此如果您了解其中一个的工作原理,您就会了解另一个。

(是我一个人,还是本章有很多“对立面”的功能?)

让我们从介绍三角形添加功能开始。我们想要在这个函数中做的第一件事是在数组中找到一个要写入的空闲三角形。(本质上,这是一个只写的顶点缓冲区,因为我们将信息放入数组中并简单地将其发送到API。)在我们拥有一个可用的三角形后,我们可以用顶点填充它的信息作为参数传递给函数的菱形信息。这里的所有都是它的!我们现在可以继续执行步骤4。

初始化 v1.00.0

初始化例程与步骤 3 中的例程没有太大区别，
并且对该功能所做的主要补充已经
在我们讨论拆分/合并优先级队列系统时谈到过。此过程的唯一其他主要补充
是我们必须
将顶级钻石放在拆分队列上，因为所有其他钻石都来自并抓取它的三角形以
开始三角形渲染
过程。为此，我们只需将此代码添加到初始的末尾
化功能：

```
pDmnd= m_pLevel1Dmnd[1][1];  
入队 (pDmnd,SPLITQ,IQMAX-1) ;  
分配三 (pDmnd,0) ;  
分配三 (pDmnd,1) ;
```

到目前为止，我们的讨论中真正缺少的就是这些
观点。到目前为止，您应该可以轻松理解其余的初始化过程或根据本章提供的信
息编写自己的代码。

更新 v1.00.0

喘气！是的，我们实际上为这一步提供了更新功能！该函数对网格进行逐帧更新。
在这个函数中，
我们要更新所有排队钻石的优先级，然后我们
想要对优先级队列中的菱形进行实际拆分/合并，直到满足以下情况之一：

- 已达到目标三角形计数或已达到准确度目标。
- 我们没有时间进行拆分/合并。（我们想限制
每帧完成的拆分/合并量。）
- 我们的钻石池中的免费/解锁钻石用完了。

在满足其中一种情况之前，我们可以拆分/合并到我们的
心满意足！尽管我们确实希望保持良好的网格水平
粗糙度，我们不希望网格太粗糙。（这是由
检查“当前值”与“最大值”，例如检查
当前三角形计数与最大三角形计数的对比。）

162 7.无论你在哪里漫游

渲染 v1.00.0

Step 4 的渲染功能很简单。使用我们之前创建的顶点缓冲区,您可以通过一次调用轻松输出所有要渲染的顶点。例如,随附的演示将所有顶点/纹理坐标输出为 OpenGL 动态顶点数组。(为简单起见,我在 CD 上的演示中只使用一种纹理,而不发送颜色信息。)但是,您可以将渲染功能移植到您选择的 API。话虽如此,demo7_4 现已开放供您查看。如果您使用 4096×4096 高度图,您会注意到此演示中最大的改进。考虑到到本章为止,我们一直在使用 512×512 高度图,我认为可以肯定地说这个算法“相当”强大。你不同意吗?

查看图 7.22 以查看演示 7_4 的运行情况 (在 CD 上的 Code\Chapter 7\demo7_4 下)。

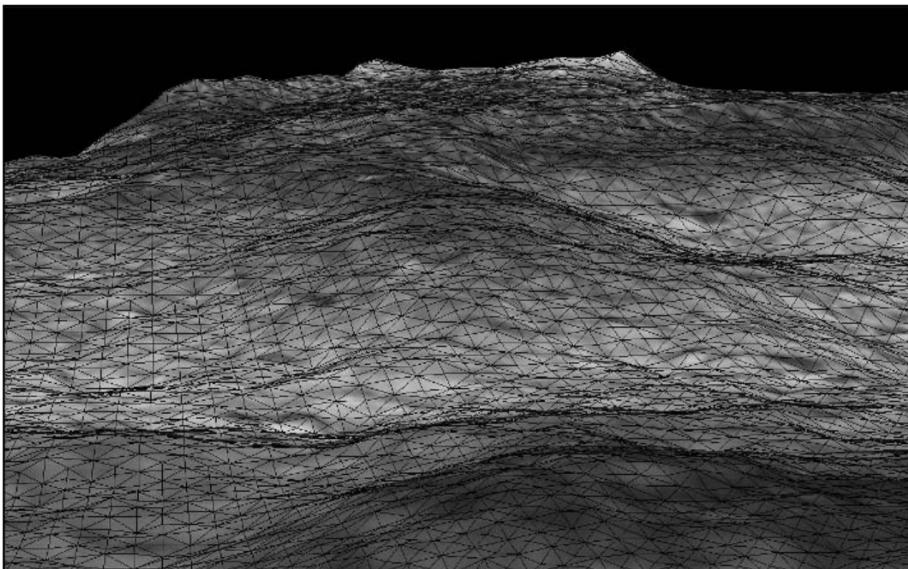


图 7.22 来自 demo7_4 (我们实现双优先队列) 的网格线框渲染,混合了网格的实体渲染。

如果你用 demo7_4 试验一段时间,你一定会发现网格有一个明显的缺陷 (见图 7.23)。这是因为引擎支持比我们的高度图分辨率支持的更高级别的细节。所以,如果你可以创建一个大的高度图 (8192×8192)

可能是一个很好的尺寸)或以某种方式动态计算高度值,以便引擎受到其允许的 LOD 的限制,而不受高度图分辨率的限制,这种“楼梯效应”将消失。

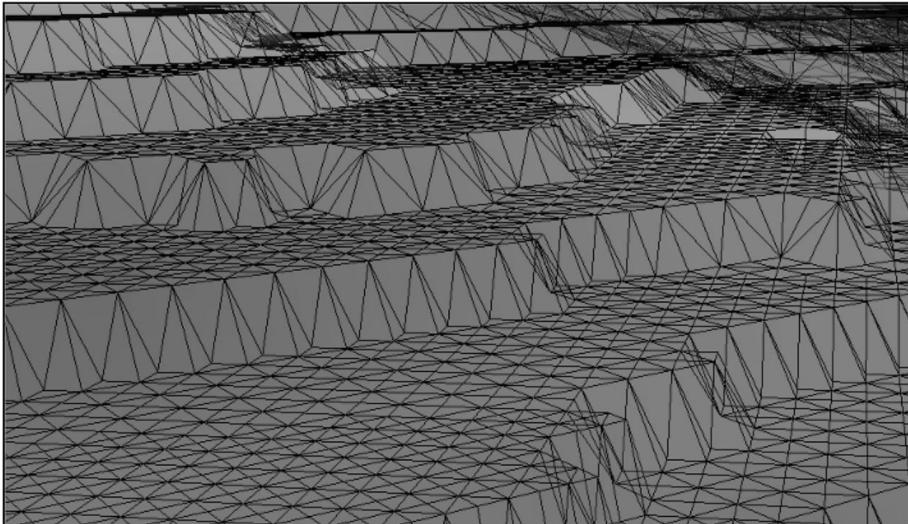


图 7.23 当 ROAM 引擎支持的细节级别超过高度图时,会出现“楼梯”多边形伪影。

概括

这是这本书的最后一章地形算法……哇,真是太棒了!在本章中,我们涵盖了许多 ROAM 信息,从几年前发布的原始论文开始,一直到算法的“新”版本。我们分四个步骤实施 ROAM 2.0 (ROAM 的新版本),从简单的多边形镶嵌开始,到一个成熟且功能强大的实施结束。下一章将是一个巨大的旅程,通过各种各样的技巧、加速和特殊效果。做好准备!

164 7.无论你在哪里漫游

参考

1 Duchaineau, M. 等人。 “ROAMing Terrain:实时优化适应网格。” IEEE 可视化 97。 81-88。 1997 年 11 月。<http://www.llnl.gov/graphics/ROAM>。

2 特纳,布莱恩。 “使用 ROAM 进行实时动态细节地形渲染。” http://www.gamasutra.com/features/20000403/turner_01.htm。

第 8 章

包装

向上：
特别的
效果及更多

166 8.总结:特效等

我们在户外的旅程即将结束,所以——以一种非常直接的方式出去。在此过程中,我们将对程序员可以使用现代硬件访问的一些最酷的特效进行详尽的演练。为了适应所有可能的效果,我将只为您提供本章的议程并开始它。以下是我们将要讨论的内容:

■渲染水的两种选择 ■使用简单的图元来
渲染场景的周围环境

■相机-地形碰撞检测和响应 ■渲染雾的两种选择 ■粒
子引擎及其与地形的使用

这是我们可以谈论的各种各样的话题。好吧,我向您保证,每个主题都
会很有趣,并且会为您提供增加户外场景真实感的选择。不过,如果我继续说
下去,我们将永远无法讨论它们,所以让我们开始吧。

一切都在水中

水渲染是任何逼真的户外场景的重要组成部分。当然,某些类型的场景——例
如沙漠场景——可能不需要一块水,但对于大多数场景来说,一小块水会极大
地增加场景的气氛。我敢肯定,你现在真正想到的问题是:“我们要做什么?”

好吧,我会告诉你的。我们将实现两种不同的水算法:一种简单的实现,另一
种稍微复杂但无限酷的实现。不再拖延,让我们开始简单的实现。

让水流动,第 1 部分

在我们第一个水渲染系统的实现中,我们将依赖一个单一的纹理分布在一个
四边形和一些简单的

具有该纹理的动画。实际上,编程就像听起来一样简单,除了在我们正确设置系统后会出现一个“棘手”的问题……但是,我走得太远了;让我们一步一步来。

让我们首先列出我们希望我们的简单水实现具有什么:

- 加载单个纹理以表示水的能力
表面
- 渲染纹理/彩色 (和 alpha 混合)的能力
四边形

是的,这是一个很小的列表,但不要害怕。我们将在本章中列出清单,并且会有许多更大的清单(我知道你是多么喜欢需要遵守的大型要求清单)。

正如您从前面的列表中看到的那样,我们没有太多工作要做才能启动并运行第一个演示。

首先,我们需要讨论我们将如何渲染我一直提到的这个四边形。看图 8.1,我们有一个简单的地形网格。

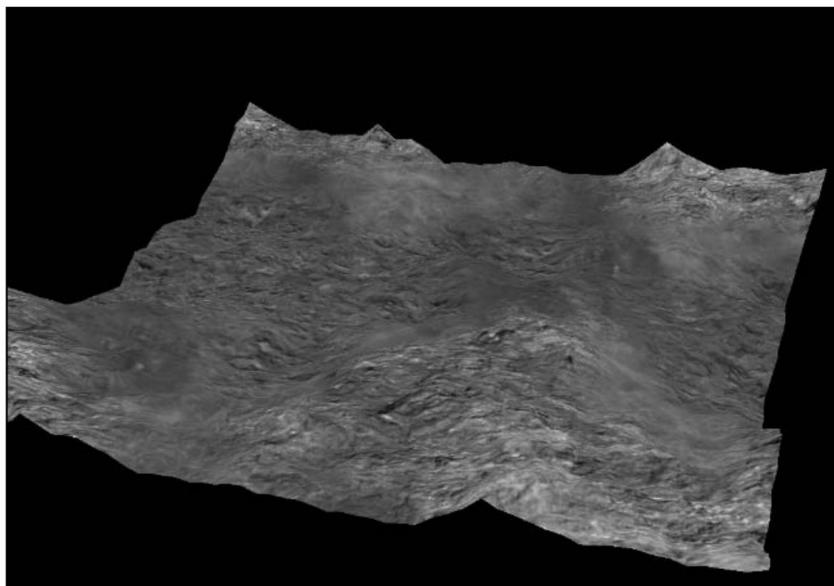


图 8.1一个简单的地形网格。

168 8. 总结·特效等

我们将选择一个合适的位置（在 Y 轴上）让我们的水块栖息，并通过网格“切割”一个四边形，如图 8.2 所示。

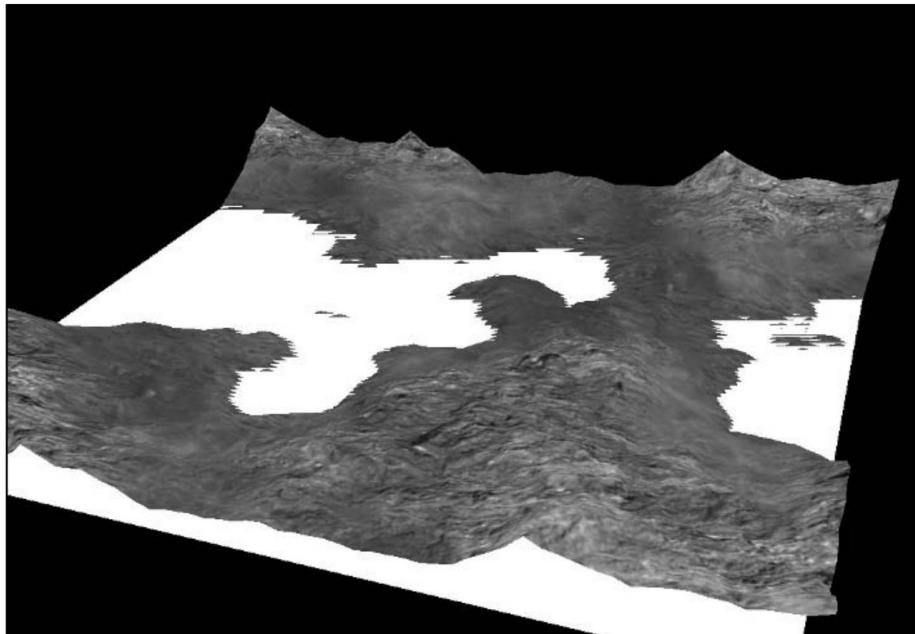


图 8.2 一个简单的地形网格，一个大的四边形将它“切片”成两半。

那是我们美丽的小四边形，很快就会成为我们的水网。注意图 8.2 中存在的撕裂量很重要，因为我们将在几段中处理它，但现在，只注意它的存在。

渲染四边形相当简单。您需要做的就是让用户设置四边形的中心，从用户那里获取补丁的大小（以世界单位为单位），然后将顶点发送到渲染 API。如果您似乎对此有疑问，请查看随附 CD 上的 demo8_1/water.cpp。

我们的四边形现在设置正确；我们现在需要做的就是添加纹理。我们在演示中使用的纹理（参见图 8.3）相当简单，但在将其应用于四边形并设置动画后，它就可以很好地发挥作用。

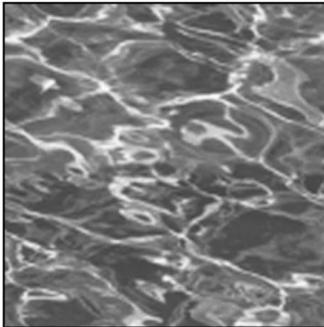


图 8.3 demo8_1 中使用的 256×256 水纹理。

将这个单一生理分布到整个补丁（特别是因为补丁可能最终会变得相当大演示中的补丁是 1024×1024 世界单位）会导致一个丑陋的水网格。我们想在网格上重复纹理几次，类似于我们对细节映射所做的。接下来，我们可以为网格添加对 alpha 混合的支持（我的意思是，真的，你最后一次看到不透明的海洋是什么时候？）以增加真实感。查看图 8.4 以了解我们目前在实现方面的立场。

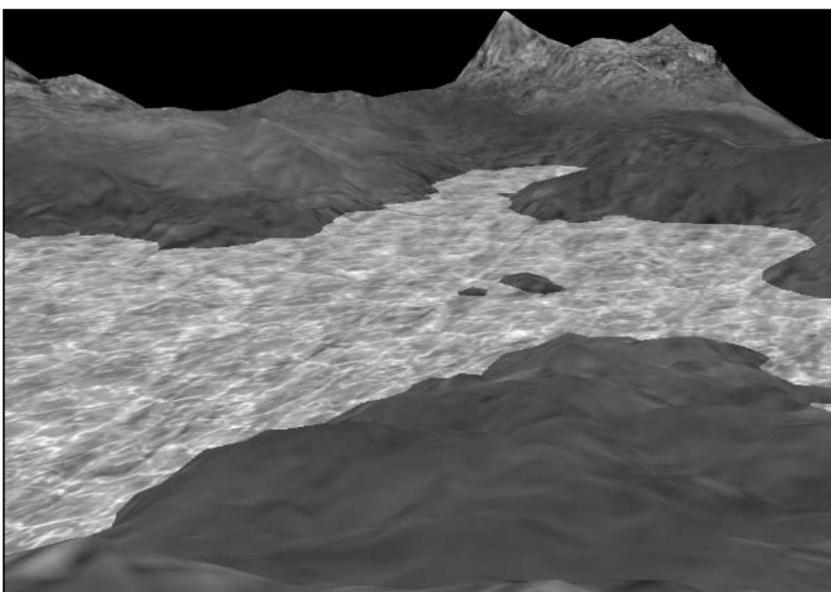


图 8.4 迄今为止的简单水实现（纹理四边形和 alpha 混合）。

170 8. 总结:特效等

在完成之前,我们需要在简单的水实现中添加最后一件事。我们需要对纹理进行“动画处理”,使其看起来比静态纹理四边形更逼真。为了完成这个“动画匹配”,我们将通过发送到渲染 API 的纹理坐标获得一些乐趣。在随附的演示中,我们要做的就是将 Y 纹理坐标每帧增加 0.1f,营造“流水”的感觉。总的来说,这是一种相当便宜的渲染水的方法,但如果你的多边形/速度预算不足,它是一个很好的选择。这就是这个简单的水实施。

去查看演示 8_1 (在 CD 上的 Code\Chapter 8\demo8_1 下),并从中获得一些乐趣。我们的下一个水实施将吹掉这个……好吧,水!

让水流动,第 2 部分我们上一个水的实现总共使用了一个由两个多边形组成的图元来渲染整片水。相比之下,我们将要讨论和实现的实现将更加多边形密集,因为我们的网格将由一系列均匀排列的多边形组成(类似于我们在进行蛮力地形渲染时使用的多边形排列)。通过使用许多多边形,我们将能够创建更逼真的水,这需要添加波浪和反射贴图,而不是我们在之前的实现中使用的静态纹理贴图。

这个实现比我们刚刚讨论的前一个要复杂一些,所以让我们再列出一个这个实现需要能够做的事情以及我们在编码时需要记住的一些事情:

- 顶点和法线缓冲区。因为水网格是动态的,我们需要实时的硬件光照作用在它上面,让它看起来很逼真。
- 顶点法线的实时更新。这样,可以实现逼真的光照(使用 API 的硬件光照)来为水增加更多“深度”。 ■顶点计算以创建一系列物理上逼真的波浪和涟漪。(从技术上讲,它在物理上并不现实,但看起来是那样的。)

■自动为水的反射贴图生成纹理坐标。这样,水看起来就像在“反射”它周围的区域。

这是这个演示的列表。它是前一个演示的议程的两倍,但它会让你想到当我们到达本章的第十二个演示时列表的大小。(是的,这一章有12个演示。)但实际上,每个演示的议程都会很短,但是,嘿,我不得不吓唬你一点!

所以,最好的起点是……嗯,一开始。具体来说,我们将创建一个高度细分的多边形网格,应用“反射贴图”(见图8.5)并操纵网格的顶点以创建一系列波浪。当我们这样做时,我们希望硬件照明能够增强水的真实感,因此我们必须为网格动态生成顶点法线。我知道这一切都很难,但我们会一步一步来。

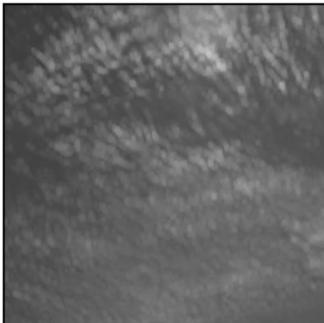


图8.5反射图示例（也用于demo8_2）。

从图8.5中可以看出(反射贴图的示例,巧合的是,我们将在本章的演示中使用相同的反射贴图),反射贴图确实没有什么特别之处。反射贴图所做的只是模拟水反射周围的环境。这听起来很简单,但你可以用它做很多很酷的事情。

例如,您可以在每一帧(或至少在视点发生变化时)将整个场景渲染为纹理,并将该图像用作水的反射贴图。这只是一个想法,但在实施中往往能得到很好的结果。然而,我们不会实施这个很酷的

172 8. 总结:特效等

本书中的技术,因为它不是一种实用的实时技术,但值得思考。(嘿,你甚至可以在我的网站上找到它的演示,<http://trent.codershq.com/>,有时!)无论如何,回到手头的话题.....

我们的顶点缓冲区的设置将类似于我们在第 2 章“地形 101”、第 3 章“纹理地形”和第 4 章“光照地形”中使用的蛮力地形引擎。我们将沿 X 轴和 Z 轴布置基本顶点,并将 Y 轴用于变量值(顶点的高度值)。X/Z 值将在整个程序中保持不变,除非您想做一些奇怪的事情,例如拉伸水网。除此之外,这些值保持不变。要创建水波纹等,我们将仅更改网格的 Y 值,这将引导我们进入下一个主题:更改顶点缓冲区的 Y 值以创建逼真的波纹和波浪。

对于我们的水网,我们有几个缓冲区。我们已经讨论了其中的两个缓冲区:顶点缓冲区和法线缓冲区。但是,我们现在要讨论的就是力缓冲器。力缓冲区包含表示作用于顶点缓冲区中某个顶点的外力量的所有信息。

查看图 8.6 以了解我们将要做什么的直观示例。

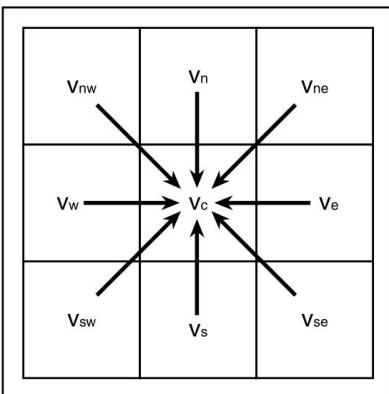


图 8.6 作用在当前顶点 V_c 上的环绕力。

图 8.6 显示了我们如何计算当前顶点的力值(图中的 V_c) ,并考虑了

作用于周围的顶点。例如,如果顶点 V 处于静止状态,并且在点 R 处产生了涟漪,则涟漪的力最终会遇到 V。(我们希望在为我们的演示创建初始涟漪后,水继续产生涟漪,所以我们假设一个波纹最终会影响网格中的每个顶点。)这会导致 V 周围的顶点,尤其是波纹方向上的顶点,影响 V,并且 V 在其他顶点停止的地方继续波纹的力.我知道,这在文本中都非常模糊。图 8.7 应该可以帮助您理解。

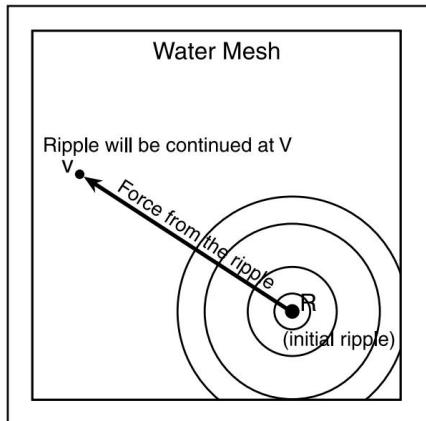


图 8.7 绑定所有顶点的涟漪（蹩脚的指环王扯掉）。

每一帧我们都将更新“力缓冲区”,它存储作用在每个水的顶点上的外力的量。

而且,对于每个顶点,我们将考虑围绕中心顶点(八个顶点)的每个顶点的力。在我们用值填充力缓冲区之后,我们必须将力施加到顶点缓冲区,然后为下一帧清除力缓冲区。(我们不希望我们的力量逐帧叠加。那看起来真的很奇怪。)

这显示在以下代码片段中:

```
对于 (x=0; x<m_iNumVertices; x++)
{
    m_pVelArray[x] += ( m_pForceArray[x]*fDelta );
```

174 8. 总结·特效等

```
m_pVertArray[x][1] += m_pVelArray[x];  
  
m_pForceArray[x] = 0.0f;  
}
```

我们在这个片段中所做的只是将当前力（在考虑了时间增量之后，以便可以实现与帧速率无关的运动）到顶点速度缓冲区（用于帧到帧的顶点速度相干性），然后将其添加到顶点的 Y 值，从而为水缓冲区设置动画。呜呼！我们现在有一个完全动画的水缓冲区，好吧，考虑到我们在网格中的某个位置开始了初始波纹：

```
m_pVertArray[rand()%(SQR(WATER_RESOLUTION))][1] = 25.0f;
```

这条线从水网格中随机位置的波纹开始，高度为 25 个世界单位。这条线开始了我们水网的所有动画。

但是，在您查看演示之前，我必须告诉您另一件事。即使我们有一个完全动画的水网格，我们还缺少一件事：逼真的照明。我们将要逐帧计算整个网格中的顶点法线，将法线发送到渲染 API，并让渲染 API 使用这些法线将硬件照明（每个顶点）添加到我们的网格中以增加我们的水模拟的真实性。只要您了解基本的 3D 理论，计算这些法线就相当简单，但记住每帧更新法线至关重要；否则，你最终会得到一些看起来很平坦的水。

这就是水！您现在可以查看 demo8_2（在 CD 上的 Code\Chapter 8\demo8_2 下），并专目睹我们新的水渲染系统令人难以置信的美感。此外，查看图 8.8 以查看演示中的静态照片。（因为我们一直在努力让水实时看起来不错，所以静态截图并不能保证效果。）

基于基元 环境 101

到现在为止，我敢打赌你真的厌倦了新的地形实现，只是为了看到在黑色之上渲染的地形

基于基元的环境 101 175

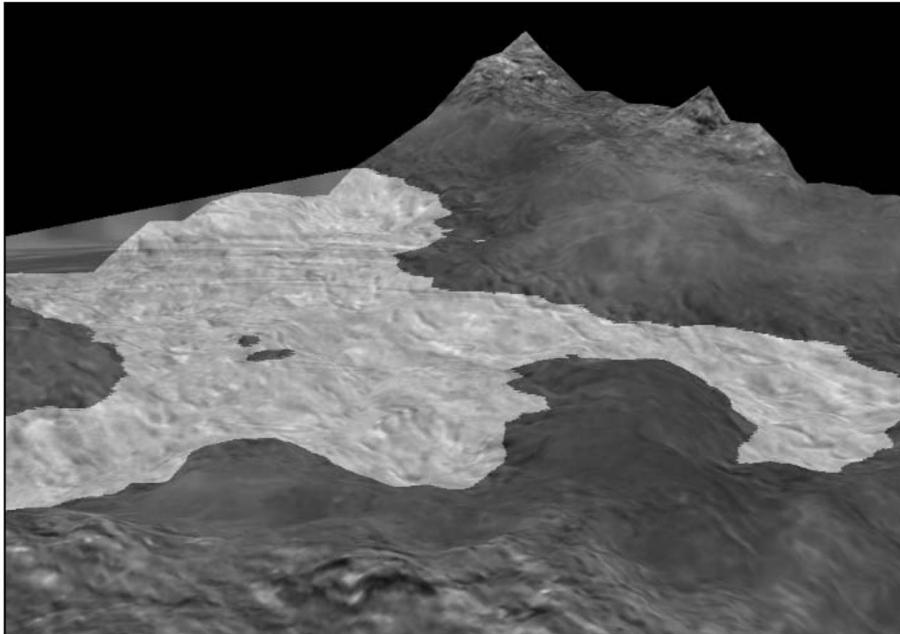


图 8.8 来自 demo8_2 的屏幕截图,显示了新的水模拟引擎。

背景。多么无聊!好吧,现在我们将努力将黑色背景制作成一个既漂亮又快速的简单环境。我们将讨论的第一种环境是通过使用天空盒来实现的。第二种类型,我个人最喜欢的,是通过使用天穹来实现的。让我们开始破解这段代码吧!

在天空盒之外思考可视化天空盒的最佳方法是复制图 8.9,将其剪掉 (你的副本,而不是书中的图……你不会想错过我要说的现在这个页面的另一边,你会吗?) ,并尝试把它变成一个立方体。 (你可以在这个练习中使用胶带。) 是的,就像你在小学做的一样!

这正是我们在本节中要做的。我们将采用六个纹理并将它们制作成一个立方体,为我们的户外场景构成周围环境。听起来很奇怪,是的,但它确实有效。不,这不是一些蹩脚的商业广告。再看图 8.9,想象一下它会是什么样子

176 8. 总结:特效等

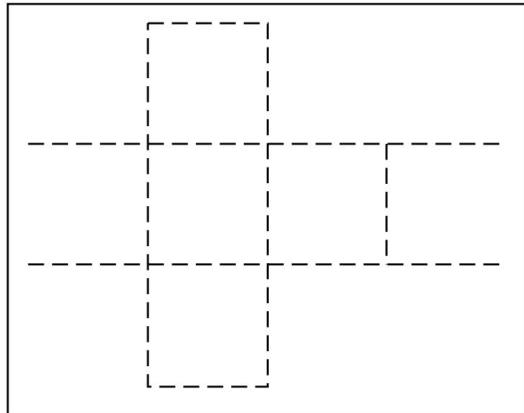


图 8.9 可用于制作简单纸立方体的剪切图案。

无缝融合在一起的纹理。 (就像魔术一样,我把你脑海中的那个图像变成了图 8.10。)到目前为止,它是对演示“黑屏环境问题”的半完美答案。

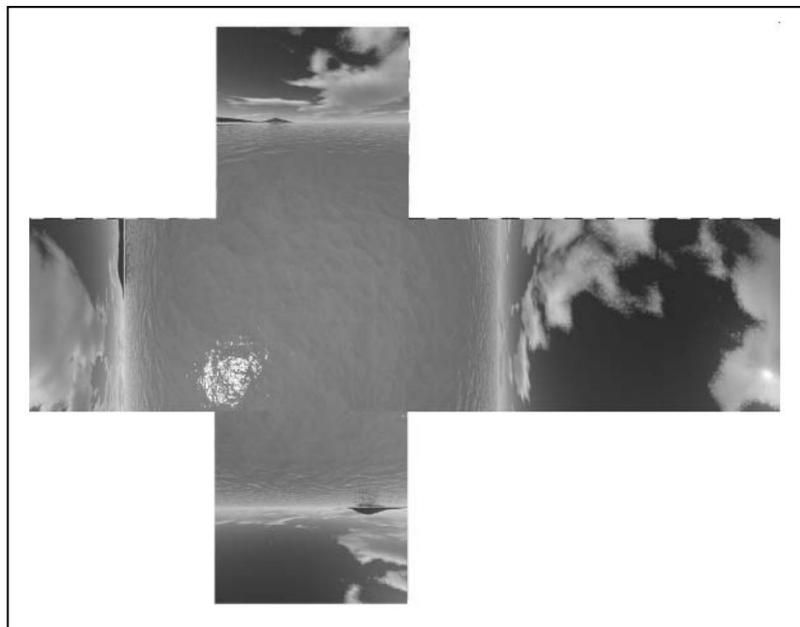


图 8.10 纸立方体,带有与 demo8_3 的天空盒一起使用的天空盒纹理。

希望你能用剪纸做一个纸立方体
 我提供给你的。现在我们需要将天空盒与
 代码,而不是我们的手和胶带。这实际上比它更容易
 声音。我们需要将这六个纹理加载到我们的程序中,构造
 六个四边形中的一个简单立方体,并将我们的六个纹理映射到相应的
 页面中。

在我们的实现中,我们希望用户提供
 天空盒及其最小和最大顶点。(这就是我们所需要的
 定义立方体,如图 8.11 所示。)除此之外,
 我们的代码可以处理实际的渲染。

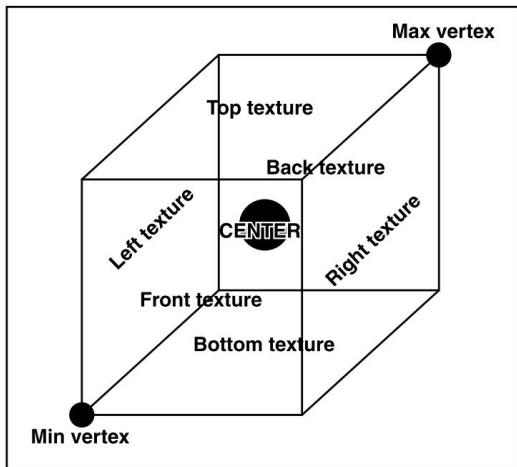


图 8.11 在给定中心、最小顶点和最大顶点时构造
 天空盒。

这就是天空盒解释的全部内容 简单而甜蜜!如果你需要
 要查看天空盒渲染的细节,请查看 skybox.cpp
 CD 上的 demo8_3 目录 (位于 Code\Chapter 8\
 演示8_3)。现在,看看 demo8_3 或图 8.12 看看什么是天空盒
 看起来像在行动。正如我所说,这是一种简单的渲染
 周边环境,但也有几个缺点。
 首先,除非天空盒使用完美的纹理,否则场景看起来有点不合适。第二个问题是没有

随机化的空间很大。天空盒纹理需要非常逼真准确才能发挥很大作用,因此分形
 生成是
 不可能的。最后,您为天空盒提供的纹理

178 8. 总结:特效等

是每次程序运行时都会使用的那些。但是,在接下来的两节中,我们将了解天空盒的一个很酷的替代方案:天空穹顶。让我们开始吧!



图 8.12 显示天空盒的屏幕截图。

住在天穹下我们将再我们的演示中使用天穹。真正的问题是:什么是天穹?嗯,就我们的目的而言,天穹是程序生成的球体(切成两半,所以我们有一个半球体)。首先,我们必须讨论如何生成圆顶以及如何为其生成纹理坐标。

圆顶生成变得简单圆顶生成对于没有扎实数学背景的人来说不是一件容易的事,所以它往往会使很多人感到困惑。如果您是其中之一,您可能想查看一篇好的天穹生成文章 1并将其用作此解释的参考(与该文章的解释相似,至少对于理论解释而言),如果它混淆了你。

基于基元的环境 101 179

无论如何,为了开始讨论,我将向您介绍描述球体的基本方程(见图 8.13),该球体位于 3D 坐标系的原点,半径为 r。

$$x^2 + y^2 + z^2 = r^2$$

图 8.13 描述半径为 r 的 3D 坐标系的以原点为中心的球体的简单方程。

笔记

为了生成一个圆顶,我们需要使用很多与我们相同的数学将用于生成球体。(毕竟,我们基本上是在生成一个球体 只是一个球体的二分之一。)因此,本节中的许多信息都与圆顶和球体的生成有关。

我们可以从前一个方程中推导出一个更适合我们目的的更简单的方程,如图 8.14 所示。这种重写允许我们计算位于球体上的点的信息。

$$f(p_c) = x^2 + y^2 + z^2 - r^2 = 0$$

图 8.14 重写了 8.13 中的方程,以便计算给定点 (P_c) 的值更容易。

但是,使用该等式计算许多点可能会有点复杂。我们希望将注意力转向球坐标系而不是笛卡尔坐标系,我们在这一代中一直在使用它。考虑到这一点,我们需要重写前面的方程并使用球坐标,如图 8.15 所示。

180 8. 总结:特效等

$$p_c = (x, y, z) = f(\Phi, \theta) = (fx(\Phi, \theta), fy(\Phi, \theta), fz(\Phi, \theta))$$

图 8.15 将 8.14 中的方程改写为与球坐标系一起使用。

在等式中,phi (ϕ) 和 theta (θ) 分别表示球面上点的纬度和经度。如果你不记得你的中学时代 (我知道我不记得), 纬度代表平行于地球赤道的线 (它们围绕球体向左/向右移动), 经度垂直于赤道 (上/下)。

考虑到所有这些,我们可以得出最终方程,它可以定义球体上任何点的值,如图 8.16 所示。

$$\begin{aligned} fx(r \sin(\Phi) \cos(\theta)) \\ f(\Phi, \theta) = fy(r \sin(\Phi) \sin(\theta)) \\ fz(r \cos(\Phi)) \end{aligned}$$

图 8.16 描述 3D 球体上任意点的最终方程 (使用球坐标系)。

该方程可用于生成整个球体,但我们只需要生成半个球体。不过,我们稍后会处理这个问题。现在,我们需要讨论如何在代码中实际使用前面的等式。首先,我们可以选择在一个球体上拥有大量的点——几乎是无限多的。事实上,我们可以选择在球体上拥有无限数量的点,但因为我们不知道真正的无限是什么,所以我倾向于坚持接近无限的答案。当然,这有点模糊,但它给我们留下了更多的正确空间!无论如何,正如我所说,我们可以为我们的球体选择几乎无限数量的点,所以很自然地假设我们将不得不设置具有某种分辨率边界的圆顶生成函数;否则,我们将拥有一个非常高度镶嵌的圆顶。

我们究竟需要做什么来生成球体,以及我们必须做什么来限制球体的顶点分辨率?好吧,这两个主题是相辅相成的,所以让我们同时讨论它们。首先, θ 可以在0到 2π 之间变化(以弧度为单位;以度为单位,可以在 0° 到 359° 之间变化)。在那个范围内,我们可以找到一个半无限的数字,所以我们要做的是找到一个很好的值来跨越这个范围。

警告

确保在编写自己的圆顶生成实现(并且使用度数)时,将所有测量值转换为弧度,然后再将它们作为参数发送给sin和cos函数,因为它们使用弧度测量值。这往往是一个通常被忽视的巨大错误。请记住:朋友不要让朋友以度数向C/C++三角函数发送参数。

例如,让我们转换为度数。在 0° 到 360° 的范围内,我们可以选择 20° 的步幅值,我们最终会得到 $360^\circ/20^\circ$ 的值,或者每列18个值顶点。您选择的步幅越小,最终得到的顶点就越多。我们可以对 ϕ 值做同样的事情。

现在我们需要将前面的文本转换为代码,令人惊讶的是更容易理解比文字。

```
for( int phi=0; phi <= ( 90-phiStride ); phi+= phiStride )
{
    for( int theta=0; theta <= ( 90-thetaStride ); theta+= thetaStride
)
    {
        //计算 phi 处的顶点,theta vertexBuffer.x=
        radius*sin( phi )*cosf( theta ); vertexBuffer.y=半径*sin(phi)*sinf(theta);
        vertexBuffer.z=半径*cos( phi );

    }
}
```

看看这有多简单?好吧,这就是为穹顶生成顶点所需要做的一切;然而,现在我们需要将这些顶点组织成一个三角形带。这意味着在生成过程中,我们从每个单独的顶点制作一个三角形(通过将该顶点连接到两个周围的顶点),然后计算纹理坐标。

182 8. 总结:特效等

这比听起来要简单得多。如果您有一两个问题,请查看 CD 上 demo8_4 目录中 skydome.cpp 中的代码。

渲染天穹生成天穹网格是最难的部分。现在我们只需要

坐下来将顶点缓冲区发送到渲染 API。不过,首先,我想讨论一下我们想要使用什么样的纹理来纹理化天穹。使用天穹最酷的部分是你不需要一堆纹理来让它看起来不错。你只需要一个!例如,在 demo8_4 中,我们使用如图 8.17 所示的纹理。



图 8.17 demo8_4 中用于天穹的纹理。

现在查看 demo8_4 并查看渲染代码。一切都很简单;我们只需将顶点和纹理缓冲区发送到渲染 API,它们就会被渲染为我们美丽的天穹。您可以在 CD 上的 Code\Chapter 8\demo8_3 下找到 demo8_4。享受演示和该演示的屏幕截图,如图 8.18 所示。

分形生成云纹理我们并不总是必须为我们的天穹使用静态纹理。我们还可以生成我们自己的云纹理,作为我们程序的预处理步骤。这是一件很酷的事情,因为它为您的程序增加了更多随机化(这总是很有趣,因为人们往往对冗余纹理感到厌倦)。但是,我们不会使用

基于基元的环境 101 183

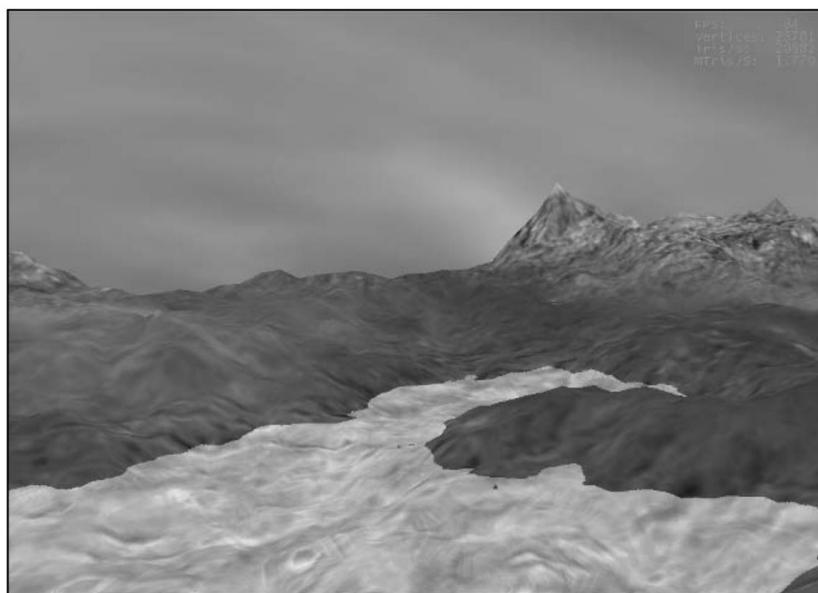


图 8.18 来自 demo8_4 的屏幕截图,我们在其中实现了一个天穹并对其应用云纹理。

我们在第 2 章中讨论过的分形生成技术;我们将使用一种新的分形生成算法。如果你理解了第 2 章中介绍的分形生成算法,那么这一节应该很容易理解。

分形布朗运动分形理论在本节中,我们将讨论分形布朗运动(FBM) 分形2。

这些是包含独特噪声函数组合的分形。我们将讨论两种类型的噪声:白噪声和粉红噪声。白噪声是随机且零星的,有点像您将电视打开到天线/卫星未接收到的频道时所看到的,并且您会看到带有烦人声音的静态屏幕。粉红噪声就像“均匀噪声”;它限制了值如何从一个点变为另一个点。

要创建粉红噪声,您需要创建一个规则的值数组,遍历数组,并在每个点存储随机值。完成后,重复它们并插入值以平滑它们;然后使噪声更加连贯和平滑。

184 8. 总结·特效等

柏林噪声

Ken Perlin 是第一个将粉红噪声用于计算机图形的人,当时他创造了他现在臭名昭著的噪声 () 功能。他甚至因此获得了奥斯卡奖。这个类型噪声现在称为Perlin 噪声。

Ken Perlin 的主页 (<http://mrl.nyu.edu/~perlin/>) 有以下很酷的小代码片段,您可以粘贴进入你的编译器并编译创建一个漂亮的小控制台程序:

```
main(k){float i,j,r,x,y=-16;while(puts(" "),y++<15)for(x=0;x++<84;putchar(":-;!/>|&IH%*#[k&15]))for(i=k=r=0;j=r*i-2+x/25,i=2*r*i+y/10,j*j+i*i<11&&k++<111;r=j);}
```

一探究竟。这是一个简洁的片段,非常值得2 程序运行所需的分钟数。

还有一个值叫做频率,可以用来控制噪声的随机性。频率越高,发生的随机性越多。(它变得更像白噪声。)

创建一些粉红噪声后,很容易开始创建通过组合或乘以其他噪声的结果的 FBM 分形功能。然而,在我们走得更远之前,重要的是请注意,您可以使用其他值来控制生成的噪声。我们已经提到了频率,但您也可以使用倍频程、幅度和 H 参数。八度值设置多少噪声值相加或相乘。幅度值_ 调整噪声值的高低(或者,如果我们使用分形生成高度图,它增加了生成值的平均高度)。H参数控制幅度的多少

每个八度的变化。这只是总结了理论这种分形生成算法。现在是实施它的时候了!

基于基元的环境 101 185

实现分形布朗
云生成运动

首先,我们将创建一个函数来获取一系列数字中的随机值,我们在第 2 章 “Terrain 101”中做了两个分形高度图生成示例。然而,这一次,我们稍微改变了这个函数。这里是:

```
浮动 CSKYDOME::RangedRandom( int x, int y )
{
    浮动 f 值;整数 n=
    x+y*57;

    n= ( n<<13 )^n;
    fValue= ( 1-( ( n*( n*n*15731+789221 )+1376312589 ) &
                2147483647)/1073741824.0f);
    返回 f 值;
}
```

这就是我们的范围随机函数。它所做的只是在一个数字范围内找到一个随机值;例如,它将找到一个介于 1 和 77 之间的随机值。这是我们实现 FBM 的基础。我们还将对该函数进行变体,称为 RangedSmoothRandom,它创建比RangedRandom函数更均匀的随机值。

在给定插值偏差后,我们还需要一个函数来插值两个值。这个函数称为CosineInterpolation,我们将把它与RangedSmoothRandom函数结合使用来形成我们噪声生成的基础。

现在为实际的噪声生成功能。我们要做的是使用RangedSmoothRandom函数计算一系列随机值。然后我们将使用CosineInterpolation对它们进行插值

功能。这是功能:

```
浮动 CSKYDOME::Noise ( 浮动 x, 浮动 y )
{
    int iX= ( int )x; int iY=
    ( int )y;浮动 f1、f2、f3、f4、fI1、
    fI2、fValue;
    浮动 fFracX= x-iX;
    浮动 fFracY= y-iY;
```

186 8.总结:特效等

```
//生成随机噪声值 f1= RangedSmoothRandom( iX,
f2= RangedSmoothRandom( iX+1, iY ); f3= );
RangedSmoothRandom( iX, f4=
RangedSmoothRandom( iX+1, iY+1 ); f1= iY+1);
CosineInterpolation( f1, f2 , fFracX ); f2=
CosineInterpolation( f3, f4, fFracX ); fValue=
CosineInterpolation( f1, f2, fFracY ); 返回 fValue;

}
```

如您所见,首先我们得到当前点的随机值;然后我们从三个周围的点获取值。接下来,我们要对前两次随机计算的值进行插值,对第三次和第四次计算的值进行插值,然后对两个插值值进行插值。砰!我们有那个点的噪声值。这个函数本身不会做很多事情,所以我们需要创建一个函数来创建整个 FBM 分形。

创建 FBM 分形值,现在我们已经有了必要的基本函数,实际上非常容易。分形创建函数需要为我们之前讨论的所有值获取参数:倍频程、幅度、频率和 H。我们还需要获取 (x, y) 坐标集,以便我们可以在特定的情况下计算 FBM 值观点。这是FBM函数:

```
浮动 CSKYDOME::FBM( 浮动 x, 浮动 y, 浮动 fOctaves,
                        浮动 fAmplitude, 浮动 fFrequency, 浮动 fH )
{
    浮动 fValue= 0;

    //使用“分形布朗运动”生成分形值
    for(int i=0;i<(fOctaves-1);i++)
    {
        fValue+= ( 噪声( x*fFrequency, y*fFrequency )*fAmplitude ); f振幅*= fH;

    }

    返回 f 值;
}
```

相机-地形碰撞检测187

看,我告诉过你这并不难!这是我们整个 FBM 生成系统,但还不足以实际创建我们的云纹理。现在我们需要处理云生成功能。云生成函数仍然需要获取分形生成的倍频程、振幅、频率和 h 信息,但它还需要一个大小参数(我们希望我们的云纹理图有多大)和一个模糊参数。(生成的分形不够模糊,无法像一团云一样通过,所以我们要稍微模糊一下。)

要创建分形数据缓冲区,我们需要这样做:

```
for( y=0; y<大小; y++ )
{
    对于 (x=0;x<大小;x++)
        fpData[( y*size )+x]= FBM( ( float )x, ( float )y, fOctaves, fAmplitude, fFrequency, fH );
}
```

这将为我们创建整个 FBM 分形;然而,我们需要做更多的事情来创建一个像样的云纹理。生成的分形将包含大量噪声,因此我们要选择一个截止高度并消除所有低于该高度的值(从而将它们变为黑色,而不是灰色阴影)。然后,我们将用蓝色替换黑色,并在整个字段上传递一个模糊滤镜。然后我们将数据发送到渲染 API 并将其转换为纹理。就是这样!我们现在有一个分形生成的云纹理!查看 CD 上的 demo8_5(位于 Code\Chapter 8\demo8_5 下)和图 8.19 以查看该演示的屏幕截图。

相机-地形碰撞 检测简单 回复

不要让标题愚弄你;这部分实际上非常简单和简短,但我敢打赌你已经厌倦了让相机直接穿过坚固的地形,所以我们将实现一些简单的碰撞检测。查看图 8.20 以了解我们需要做什么。

188 8. 总结:特效等

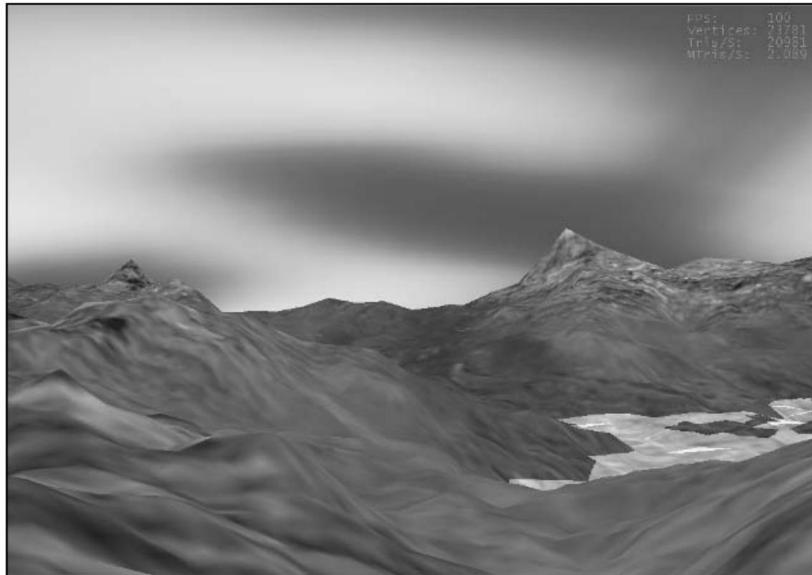


图 8.19 来自 demo8_5 的截图, 它实现了分形云纹理生成系统并使用天穹上的纹理。

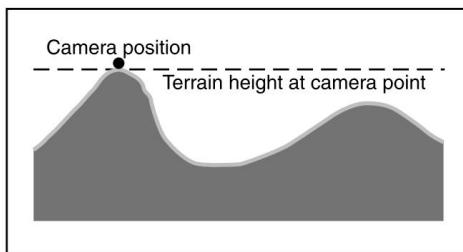


图 8.20 相机与地形碰撞示例。

必须防止相机在其当前点越过虚线, 这意味着相机不会穿过其当前点的地形。

我们必须将图中显示的内容实现到我们的代码中。这样做需要大约六行简单的代码。首先, 我想在地形网格周围添加边界, 这样相机就不再超出网格区域。我们将把相机的位置钳制到 [0, 地形网格的大小] 来实现这一点。然后我们要测试相机的 Y 坐标与相机 (x, z) 位置处地形网格的高度。

```
ucHeight= g_ROAM.GetTrueHeightAtPoint(g_camera.m_vecEyePos[0],
                                         g_camera.m_vecEyePos[2]);
```

```
如果 (g_camera.m_vecEyePos[1]< (ucHeight+5) )
    g_camera.m_vecEyePos[1]=ucHeight+5;
```

那就是整个碰撞检测和响应代码。（我还给了相机一个大约 5 像素的“空闲缓冲区”，这样我们的近剪裁平面就不会干扰地形。）如果我们的相机在那个点低于地形的高度，那么我们设置相机的高度到地形，这可以防止相机下降并穿过地形。这就是这个小提示。在 CD 上的 Code\Chapter 8\demo8_6 下查看 demo8_6。

迷失在迷雾中

在本节中，我们将实现两种类型的雾：基于距离的雾和基于顶点的雾。这两种类型的雾都有硬件支持，所以这并不是一个复杂的话题。它更多的是正确使用雾的操作指南，并展示了一些可以用它完成的很酷的事情。让我们从基于距离的雾解释开始。

基于距离的雾 基于距离的雾始终基于视点。使用它

所需要做的就是将起始深度和结束深度传递给硬件 API，然后执行任何您想要自定义雾的其他操作（颜色、密度等）。这远不是一个复杂的讨论，但是让雾看起来正确需要对系统有一点了解；否则，你最终会在你的场景中看到一些看起来很奇怪的雾。看图 8.21，看看雾是如何工作的。

如图所示，起始深度控制雾进入场景的入口点。然后雾变得越来越稠密（基于您传递给渲染 API 的密度），直到它到达结束深度，在那里它不再存在。

基于距离的雾 非常适合掩盖距离中的 LOD 变化或掩盖低 LOD。如果您想通过消除远处的物体来减少多边形数量并且您希望雾覆盖它，它也很有效。这种雾在设置上也很不错。

190 8. 总结:特效等

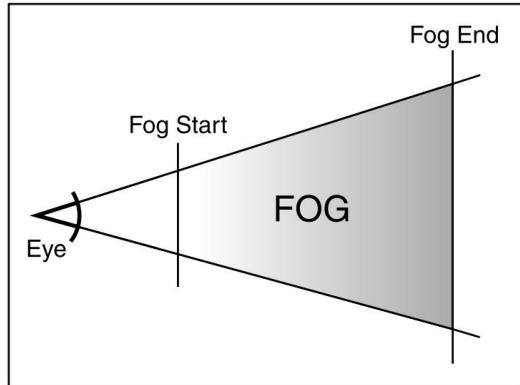


图 8.21 基于距离的雾将如何工作。
它从起始深度开始,逐渐变稠,直到达到结束深度。

演示的心情。然而,在大多数情况下,就雾实现而言,基于距离的雾并不是最佳选择。

尽管如此,这是一个相当酷且易于实现的效果。话虽如此,请查看 CD 上
Code\Chapter 8\demo8_7 下的 demo8_7 以及图 8.22 中演示的现在常规屏
幕截图。

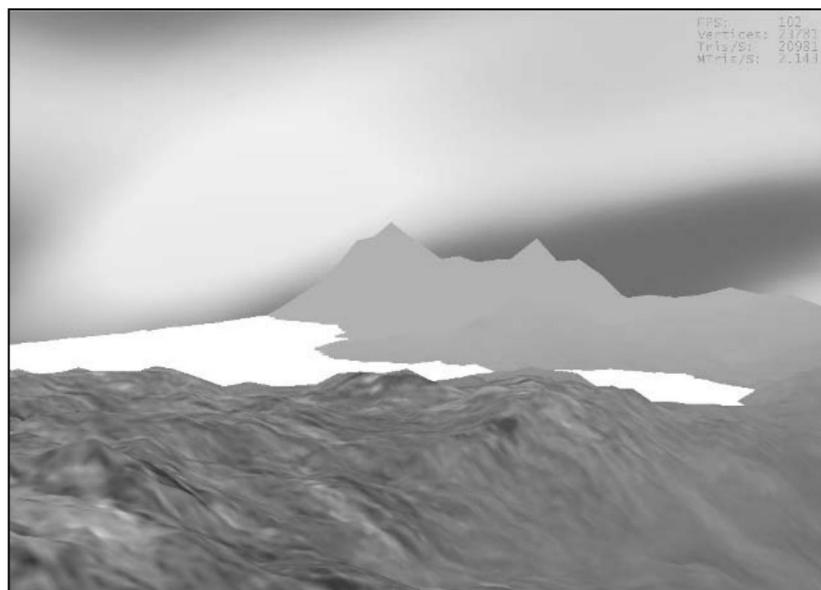


图 8.22 来自 demo8_7 的屏幕截图,其中实现了基于距离的雾。

基于顶点的雾 基于顶点的雾的 使用与基于距离的雾完全不同。

当您指定开始/结束深度时,它现在适用于 Y 轴而不是 Z 轴。您还为正在渲染的当前顶点提供雾坐标。(顺便说一句,所有这些都可以通过 OpenGL 扩展

“GL_FOG_COORDINATE_EXT”访问,这就是本节随附的演示中使用的内容。)基于顶点的雾,有时称为体积雾,也不依赖于视图。

在您提供坐标和开始/结束高度后,雾会在整个程序中停留在该位置,这使其成为一些很酷的效果的理想选择,例如从我们的水斑中产生雾气。

事实上,从我们的水域制造雾气听起来是个不错的主意,所以让我们继续之前的思路。我们想要做的是提供一个合适的开始/结束高度,这样我们的雾将上升到适当的高度。您需要查看此演示的源代码,因为很多代码都是特定于 API 的,但我想我会让您了解这个效果可以实现什么。

我为这个效果提供了两个演示来展示你可以用它做什么。第一个演示 demo8_8a 展示了一些简单的灰色雾气从我们的水中升起。(您可以使用 +/- 键调整雾升起的低/高。)第二个演示 demo8_8b 基于黑暗场景,从水中升起蓝色雾气,使水看起来被照亮。

我想你会喜欢这两个演示。检查它们,并查看图 8.23。

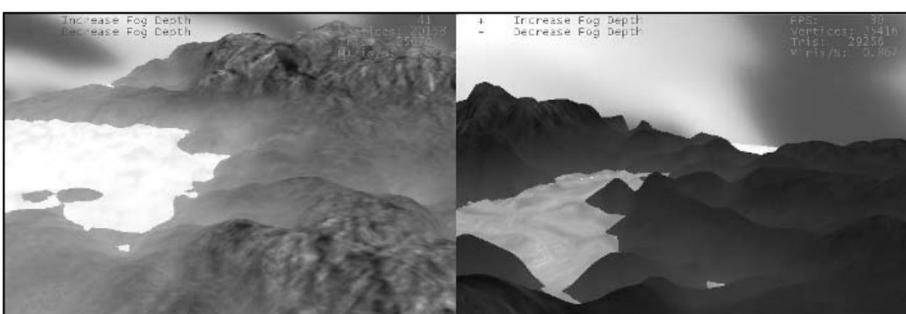


图 8.23 demo8_8a (左)和 demo8_8b (右)的屏幕截图,显示了基于顶点的
雾。

192 8. 总结:特效等

粒子引擎及其 户外应用

我知道我知道。您一定会问自己，“粒子引擎与户外场景有什么关系？”嗯，实际上，相当多。粒子引擎非常适合用于渲染雨、雪、从天而降的彗星以及地形表面的大坑。（如果你明白我的意思，你可能会在我的网站上找到一个完全一样的演示。）不过，在我们了解真正酷的东西之前，我们需要花几页关于粒子理论……主要是因为我有一个奇怪的对粒子引擎的痴迷，也是因为我认为这个理论会对你有用。

粒子引擎：基础知识如果您不熟悉粒子引擎（在这种情况下，您可能不太了解本节的介绍，因此您可能需要在了解粒子引擎后重新阅读），我会给出你是一个快速的介绍。William T. Reeves 于 1982 年开发了粒子引擎。Reeves 正在寻找一种方法来动态渲染“模糊”对象，例如火和爆炸。（粒子引擎是 pyromaniac 最好的朋友。）Reeves 提出了实现这种“模糊”渲染系统的一系列要求：

- 必须生成新粒子并将其放入当前粒子引擎。
- 必须为每个新粒子分配其独特的属性。 ■ 任何超过其寿命的粒子都必须声明“死的。”
- 当前粒子必须根据它们的脚本移动。 ■ 必须渲染当前的“活动”粒子。

我希望您注意，因为我们将实施该列表中的所有内容！不过，这并不能真正帮助您理解粒子引擎是什么。基本上，粒子引擎是一个或多个粒子系统的“管理者”。粒子系统是粒子发射器（共享基础粒子系统的属性）创建的多个单独粒子（可以是点、线、3D 模型或其他任何东西）的管理器。在图 8.24 中直观地看到这种关系。

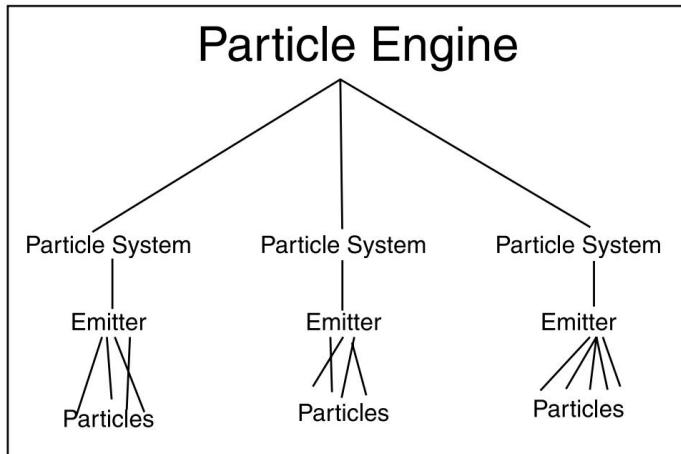


图 8.24 粒子引擎中关系的可视化解释。

图 8.24 展示了一个完整的、复杂的粒子引擎的特点,但我们将坚持本节的基础知识

因为我们不需要所有的高级功能。但是,如果您想查看复杂的粒子引擎,请查看我的网站:

<http://trent.codershq.com>,在那里你可以找到一个相当的代码复杂的引擎。

无论如何,这里有一个单个粒子的快速小“粒子时间线”。

粒子是通过迭代静态分配的缓冲区来创建的

并寻找一个“空位”(标有“死”旗),我们

可以放置新粒子。这个粒子将采用粒子系统的默认属性,并且粒子将遵循一个轨迹

直到它的生命计数器归零。然后粒子将被标记为

“死了”,整个过程将重新开始。那是单个粒子的寿命。我们可能有大约 10,000

架这样的飞行

屏幕周围。

当然,一个粒子不仅仅是一个斑点。它有很多

您可以使用这些信息来创建几乎任何可以想象的特殊效果。以下是典型粒子需要具备的属性:

- **寿命。**这就是粒子的寿命。

- **当前位置。**这是粒子的当前位置

- 2D/3D 空间。

194 8. 总结:特效等

■速度。这是粒子的方向和速度。 ■质量。用于精确模拟粒子运动。 ■颜色。这是粒子的当前颜色（RGB 三元组）。 ■半透明。这是粒子的当前 alpha 值（透明度）。 ■大小。这是粒子的视觉尺寸。 ■空气阻力。这是粒子对摩擦的敏感性

在空中。

考虑到这一点，我们可以创建一个简单的粒子结构以用于我们的粒子引擎。请记住：我们通过删除粒子系统和复杂的粒子发射器中间件大大简化了前面讨论的整个粒子引擎架构。下面的粒子结构看起来与我们将用于演示的粒子结构相似。

结构体

```
{
    浮动 m_fLife;
    CVECTOR m_vecPosition;
    CVECTOR m_vecVelocity;

    浮动 m_fMass;
    浮动 m_fSize;

    CVECTOR m_vecColor;浮动 m_f 半透明;
    浮动 m_fFriction;
};
```

如您所见，我们将之前的粒子属性列表中的所有要求都实现到了粒子结构中。我们将如何使用它？好吧，我将向您展示，但首先让我演示粒子引擎类，因为如果我先向您展示它会更容易理解所有内容：

CPARTICLE_ENGINE 类

```
{
```

粒子引擎及其户外应用195

私人的：

```
SPARTICLE* m_pParticles; int
m_iNumParticles;

int m_iNumParticlesOnScreen;

//重力
CVECTOR m_vecForces;

//基础粒子属性 float m_fLife;

CVECTOR m_vecPosition;

浮动 m_fMass;
浮动 m_fSize;

CVECTOR m_vecColor;

浮动 m_fFriction;
```

无效 CreateParticle (浮动 fVelX, 浮动 fVelY, 浮动 fVelZ) ;

上市：

```
CPARTICLE_ENGINE (无效)
{}
~CPARTICLE_ENGINE (无效)
{}
};
```

这就是我们现在的引擎结构（减去成员自定义函数，但这些都是您之前见过的一行“Set_____”函数，即在CTERRAIN类中）。正如你所看到的，这个结构有很多与我们的粒子结构相同的变量。这些是我们在初始化时赋予每个粒子的“父级”或默认值。我们还有一个潜在的粒子缓冲区（它将在初始化函数中创建，现在是我们代码的常规方面）。我现在想专注于CreateParticle，因为它是我们的引擎的一个组成部分。

196 8. 总结:特效等

CreateParticle函数实现了我们之前讨论过的“粒子生命周期”部分。它循环遍历引擎的整个粒子缓冲区,找到一个“死”粒子,然后使用该空闲空间根据引擎提供的默认值创建一个新粒子。这是一个非常简单的函数,因此您可以自己轻松编写代码。如果您遇到困难,请查看 CD 上 demo8_9 目录下的particle.cpp。

我要讨论的另一个功能是引擎的更新功能。

如果您想在第三个演示中理解它的更复杂的形式,那么理解它的简单形式很重要。此函数迭代整个粒子缓冲区,根据粒子当前的速度移动粒子,并由于空气阻力/摩擦力和作用在粒子上的外力(重力、风等)而降低/增加速度。随着粒子老化和接近死亡,我们还增加了粒子的半透明度。您可以在下面的代码片段中看到所有这些:

无效 CPARTICLE_ENGINE::更新 (无效){

```

CVECTOR 向量动量;
诠释我;

//遍历粒子
for(i=0;i<m_iNumParticles;i++)
{
    //老化粒子
    m_pParticles[i].m_fLife-= 1;

    //仅在粒子存活时更新粒子 if( m_pParticles[i].m_fLife>0.0f )
    {

        vecMomentum= m_pParticles[i].m_vecVelocity *
                    m_pParticles[i].m_fMass;

        //更新粒子的位置 m_pParticles[i].m_vecPosition+=
        vecMomentum;

        //设置粒子的透明度 (基于它的年龄) m_pParticles[i].m_fTranslucency=
        m_pParticles[i].m_fLife /
                    m_f生活;
    }
}

```

粒子引擎及其户外应用197

```
//现在是外力付出代价的时候了  
m_pParticles[i].m_vecVelocity*=1-  
m_pParticles[i].m_fFriction;  
  
m_pParticles[i].m_vecVelocity+= m_vecForces;  
}  
}  
}
```

关于第一个演示的总结。所有的初始化和关闭功能都与您在之前的演示中看到的类似，

所以你在理解它们时应该没有问题。渲染功能现在非常简单。我想坚持基础

对于第一个演示，因此演示仅呈现简单的 alpha 混合像素。查看演示 demo8_9（在 CD 上的 Code\Chapter 8\ demo8_9，并查看演示的控件（参见表 8.1）。

这可能是整本书中的第一次,我没有展示

此演示的屏幕截图;查看非移动像素的屏幕截图是没那么有趣。但是,演示非常酷。

表 8.1 demo8_9 的控件

钥匙	功能
逃生/Q	退出程序
在	以线框模式渲染
和	以实体/填充模式渲染
小号	

将粒子带到一个新的维度

好吧,我们只是将粒子带到一个新的维度;在之前的演示中,我们使用像素,现在我们将使用2D纹理。然而,使用纹理带来了一个相当严肃的新的

问题。我们在 3D 世界中制作粒子，因为

198 8. 总结·特效等

粒子基于 2D 纹理,这意味着未定义一维。因此,当观众在粒子周围走动时,他会看到纹理变得“平坦”。这是不可接受的,所以我们需要实现一个叫做广告牌的东西。

广告牌广告牌是当您需

要更改二维对象(例如四边形)的方向以使其面向用户时。为此,您需要从渲染 API 中提取当前矩阵,并根据该矩阵找到“上”和“右”向量。这对你来说可能意义不大,所以让我详细说明一下。当您从 API 中提取矩阵时,您可以将其放入一个 16 位浮点值的数组中。(这是 OpenGL 对 4×4 矩阵执行此操作的方式)。用矩阵中的值填充数组后,您可以提取上向量和右向量的信息,如图 8.25 所示。

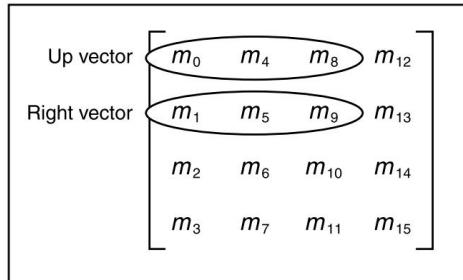


图 8.25 如何从 4×4 矩阵中提取上向量和右向量的信息。

现在我们需要将它应用到我们的四渲染代码:

```

QuadTopRight= ( ( RightVector+UpVector )
                * 粒子大小)+粒子位置;
QuadTopLeft= ( ( UpVector-RightVector )*
                * 粒子大小)+粒子位置;
QuadBottomRight= ( ( RightVector-UpVector )*
                    * 粒子大小)+粒子位置;
QuadBottomLeft= ( ( RightVector+UpVector )*
                  -粒子大小)+粒子位置;
    
```

粒子引擎及其户外应用199

然后将这些顶点发送到渲染 API (当然,带有纹理坐标)和 BAM !你现在有了广告牌粒子!

看看 demo8_10 (在 CD 上的 Code\Chapter 8\demo8_10 下) ,
并查看图 8.26。看看有多少纹理可以添加到
模拟?



图 8.26来自 demo8_10 的截图。

添加数据插值

我们将对粒子引擎进行的最后一个主要更改是数据插值。这是对我们的粒子引擎的一个很好的补充,因为它允许我们使我们想要的每一个效果都比它更真实

会一直使用上一节中的引擎。数据插值,正如你从 “Fractal Brownian Motion Fractal Theory”中知道的那样

部分,是我们在给定偏差的情况下对两条数据进行插值。对于我们的
粒子引擎,我们将使用线性插值,但可以添加二次插值以获得更酷的效果。

我们的粒子引擎,而不是跟踪单个默认值
对于某个粒子属性,现在跟踪默认启动

200 8. 总结:特效等

值和默认结束值。我们还为所有粒子属性添加了“插值计数器”。此计数器在创建粒子后计算，并将用于增加/减少当前粒子属性的值。下面是我们计算粒子大小的计数器的方法：

```
粒子[i].m_vecSize= m_vecStartSize;粒子  
[i].m_vecSizeCounter= (m_vecEndSize-particles[i].m_vecSize) / 粒子[i].m_fLife;
```

这就是我们开始粒子及其数据的方式。每一帧，我们都会这样做：

```
粒子[i].m_vecSize+=粒子[i].m_vecSizeCounter;
```

这里的所有都是它的。当然，您必须将这些概念应用于每个粒子属性，但您会明白其中的要点。请随意查看 CD 上 Code\Chapter 8\demo8_11 下的 demo8_11，它与前面的几个粒子演示略有不同，如图 8.27 所示。

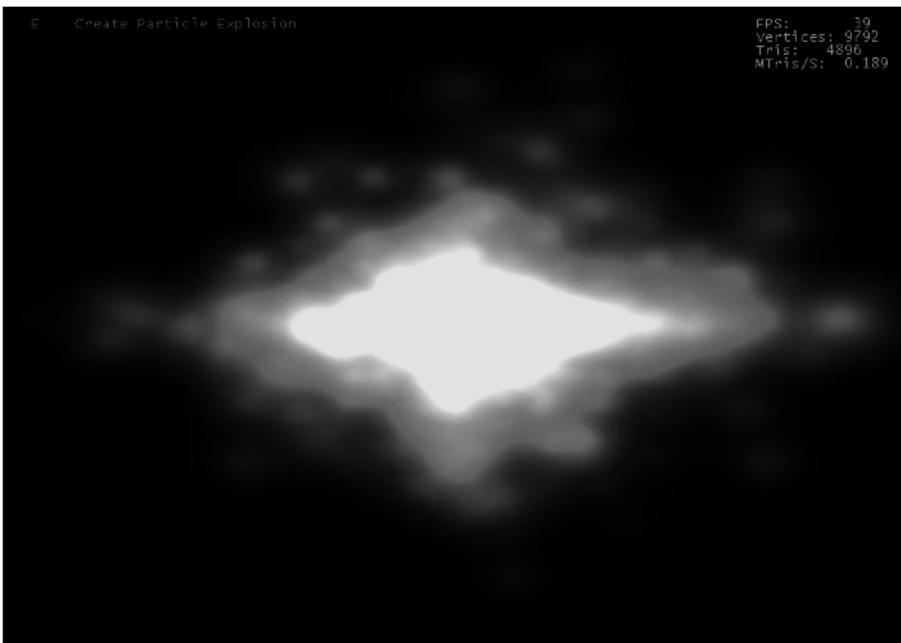


图 8.27 来自 demo8_11 的屏幕截图。

粒子引擎及其户外应用201

将粒子引擎应用于户外场景

现在,对于本章的最后一个演示,我们将在室外场景中应用粒子引擎。为此,我们将创造雨水。你有几个选项可以做到这一点,但我们要做的是在相机的眼睛位置周围创建一个假想的立方体。然后我们将在随机(x, z)坐标处用最大高度的雨滴填充该立方体。

您可能会认为我们需要一个特殊的纹理来创建雨粒子,但事实并非如此。我们需要做的是将我们的粒子大小的X坐标缩小一点,这使得我们旧的耀斑纹理变成类似于雨滴的东西(见图8.28)。

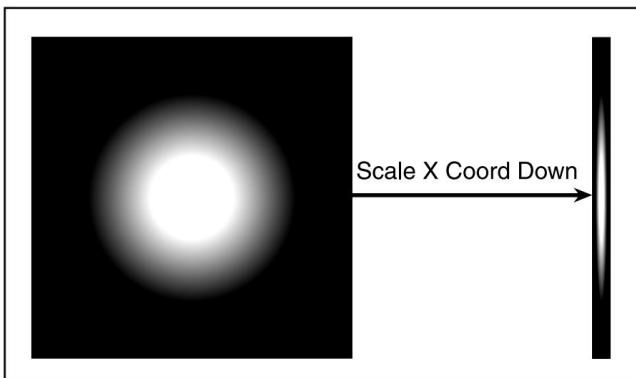


图8.28在X轴上向下缩放粒子“耀斑”纹理以创建类似雨滴的纹理。

我们的“雨立方”方法唯一不切实际的部分是它不检查与地形的碰撞,当相机被压在山上时往往会产生奇怪的效果;然而观众仍然可以看到远处的雨。这很容易通过在粒子引擎中添加碰撞检测来解决,但这是我留给你的任务。去看看本书的最后一个演示,CD上Code\Chapter 8\demo8_12下的demo8_12,以及图8.29,那个演示的屏幕截图。

202 8. 总结:特效等

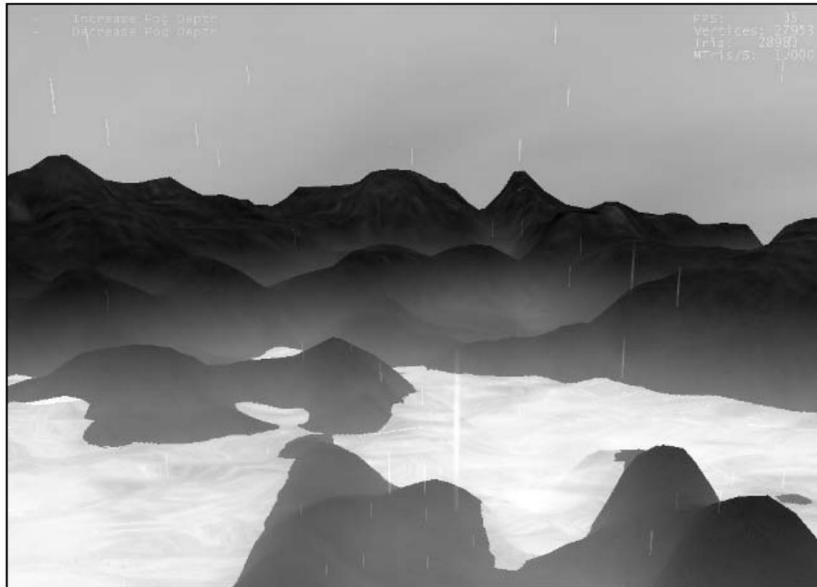


图 8.29 demo8_12 的截图,本书的最后一个演示,其中粒子引擎用于创建实时天气(雨)。

概括

本章是大量特效和技巧的恶毒贯穿。我们涵盖了水、使用天空盒和天空穹顶渲染环境、相机-地形碰撞检测、雾,以及如何将粒子引擎应用于户外场景。这也是本书的最后一章,所以不幸的是,是时候把所有的东西都收起来了。

尾声哇!想想这已

经是结束了。好吧,说实话,感觉就像结束了,我准备好好休息一下(整整两天左右)。虽然这本书与当今市场上的许多编程书籍相比相当小,但让我告诉你,这个小家伙做了很多工作。我尽我最大的努力确保本书中的所有信息都完全正确,我甚至找到了三种主要地形算法的作者(de Boer 的 geomipmap ping 算法、Rottger 的四叉树算法和 Duchaineau 的

ROAM 算法)查看章节以确保信息正确。我希望您喜欢这本书,但在我永久告别之前,让我再次向您推荐一些不错的地形信息站点。

正如我在第 1 章 “户外之旅”中提到的,虚拟地形项目是 Internet 上主要的地形信息来源之一,可以在 <http://www.vterrain.org/> 上找到它。

GameDev.net (<http://www.gamedev.net>) 有一些很好的地形文章,但更重要的是,它有一个很好的论坛,您可以在其中发布您可能遇到的任何地形问题/问题。

Flipcode (<http://www.flipcode.com>) 也有一些有用的地形教程,您可以查阅;在您启动并运行 “l337”地形引擎后,您可以将其作为“当天的图像”交给他们!

至此,本书告一段落。请务必查看随附的 CD (以及涵盖它的附录)以及我之前列出的站点。而且,我在本书中的最后一点是,不要让一套实现限制了你的想象力;始终追求创新而不是模仿。话虽如此,我离开这里是为了睡几个月的觉。祝大家编码愉快!

参考

1 Sempé, Luis R. “天空穹顶”。2001 年 10 月。
<http://www.spheregames.com/files/SkyDomesPDF.zip>。

2 Laeuchli,杰西。“编程分形。”游戏编程宝石2。
马萨诸塞州辛厄姆:查尔斯河媒体,2001. 239–246。

附录

什么是
在
光盘

206光盘上有什么

本书随附的 CD-ROM 包含许多很酷的东西。我花了许多时间编写了一系列的演示和算法白皮书放到这张 CD 上,所以请务必查看所有内容。

图形用户界面

CD-ROM 的图形用户界面 (GUI) 基于 HTML。它使您可以快速轻松地访问 CD 的各种项目和功能。大多数流行的 Web 浏览器都可以查看此 GUI,但最好的办法是使用 Netscape 4.0 (或更高版本) 或 Internet Explorer 4.0 (或更高版本) 查看它。如果我自己这么说的话,它是一个非常简洁的 GUI,所以请检查它并享受它所呈现的纯 GUI 优点。

系统要求

这张 CD 附带的演示是我关注的系统要求,所以请记住这一点。这些演示不是旧的 Pentium 1 75MHz 可以运行的任何东西,但它们也不需要最先进的计算机。以下是本书演示所需的最低要求:

- 中央处理器。需要一个 450MHz 的处理器。
- 内存。至少需要 64MB 的 RAM,但需要 128MB
推荐使用内存。
- 显卡。具有至少 16MB RAM 的视频卡是
需要运行这些演示。不过,我强烈推荐 32MB Geforce 2 (或同等版
本) 或更高版本。
- CD-ROM、DVD-ROM、CD-R、CD-RW 或 DVD-
RW 驱动器。任何
需要其中之一。(您还希望如何将 CD 放入计算机?)
- 硬盘驱动器。
要将 CD 中的所有内容复制到硬盘驱动器上,至少需要大约 125MB 的可
用空间,但这实际上是不需要的。对于只想复制算法白皮书和本书的演示/代码
文件的用户,需要大约 50MB 的可用空间,另外还需要更多空间用于您想要
安装的任何随附演示。

安装

我想我应该在介绍 CD 的基本细节之前把这部分放在前面。这样,如果您愿意,您可以跳过它们并立即查看 CD。

这张 CD 的安装非常简单。如果您使用的是 Windows 95 或更高版本并且启用了 CD 自动运行功能,CD 的菜单应该会在您的面前弹出,您可以开始探索 CD!如果菜单没有弹出,那么您需要手动调出它。为此,请转到我的电脑并双击(或单击,具体取决于您的计算机设置)CD 的图标。如果这仍然不起作用,您需要进入 CD 目录并双击(或单击)start_here.htm 文件。

从此时起,您应该能够在菜单中导航。如果您在使用 CD 时遇到任何其他问题,请随时通过 trent@codershq.com 给我发送电子邮件。

结构

CD 包含三个主要文件夹,您可以从中访问 CD 的主要组件。这些文件夹如下:

- 算法白皮书。在这些地方,您可以找到本书中介绍的三种主要地形算法背后的官方文档。还有一个关于纹理生成的教程和对原始 ROAM 算法的深入分析。 ■ 代码。在这里您可以找到本书各章中的演示和代码。所有的演示都包括一个预构建的 EXE、所有必要的文件和一个 Microsoft Visual C++ 6.0 工作区,以便您可以快速将演示加载到 VC++ 中并进行编译。 ■ 演示。该目录包含五个演示。其中两个演示展示了某种地形实现(体素引擎和分块 LOD 引擎)。还有三个演示展示了如何在游戏中使用地形引擎。其中一款游戏是令人印象深刻的 TreadMarks。此外,如果您的计算机上没有能够读取 TGA 和 RAW 文件的 Paint 程序,您将找到 Paint Shop Pro 7 的演示。帮自己一个忙,看看这些演示!

指数

数字

二维数据结构,Roettger 的四叉树
算法,106-108
3D 距离公式,geomipmapping,90-92
3D 地形、游戏开发,5-7

一个

AABB (Axis-Aligned Bounding Box) ,99-100 抽象类,已定义,19
添加/删除功能,160
空气阻力,粒子属性,194 算法蛮力,24-27

CLOD (连续细节) ,5 断层形成,27-33

中点位移,33-37
漫游,6, 128-164
Roettger 四叉树, 78, 106-126 屏幕像素确定, 91-92
分配函数,160
alpha 混合,水渲染,169 个替代演示,用于

书,8

动画,水渲染,170
API,顶点缓冲区渲染优势,81-82

应用程序,地形编程用途,4-5

参数,插值函数,50
Axis-Aligned Bounding Box (AABB),平截头体剔除,99-100

乙

骨干数据结构,ROAM
算法,149-156 基本代码,书中使用的约定,8-10
基节点
改进,133-134 嵌入,131-132

广告牌,粒子引擎,198-199 二叉三角树合并队列,132-133 节点改进,133-134
ROAM 算法,129-132 拆分队列,132-133 细分级别,129-131

边界,瓷砖,51-52

边界球,截锥体剔除,145-146 蛮力算法,高度图,24-27

缓冲区

BinTriNode 结构,134-135
数据,22-24
钻石池,153-154 临时,30 顶点,81-82

水渲染,172-174

C

C++ 风格的文件 I/O,与 C 风格的文件 I/O,21-22
相机-地形碰撞检测,187-189
光盘
内容结构, 207
文件路径约定,8
图形用户界面,206
安装, 207
系统要求,206

210指數

子节点	开裂避免方 法,83-86 定义,82 if-else 语句测试, 84-85
二叉三角树节点链接,134	
Roettger 的四叉树算法,110-111 类	
摘要 ,19	CreateParticle 函数,195-196
CQUADTREE,116-117	C-Style File I/O,作者推荐,21-22
CTERRAIN,18-21	当前位置,粒子属性,193
geomipmapping,88	
CLOD。查看连续的详细程度	
算法 云 纹理	
分形布朗运动 (FBM) 理论,183-187 天穹, 182-187	heightmap 数据加载,23 heightmap
代码,书中使用的约定,8-10	内存分配,22
碰撞检测	空指针,23-24
相机地形,187-189 缩放函数,21	数据插值,粒子引擎,199-200 deBoer,Willem H., geomipmapping 开发者,79 演示代码,书中使用的约定, 8-10
颜色	演示
高程表示,17-18 光源,59-62 粒子属性,194	替补,8
RGB 值提取,48-49 个组件,书籍部分, 11-13 个连续细节级别 (CLOD) 算法优势,77-78 个破解,82-86 个定义,76 个在需要的地方添加细 节,77-78 个缺点,78 个游戏开发, 5 geomipmapping 维护,89-92 geomipmapping 理论,79-86 多边形剔除, 78 屏幕像素确定算法,91-92	光盘,206
	分组,8 个主要, 8-10 个随机,8 个 系统要求,206 个 定义的细节贴图,52 个硬件多纹理, 54-55 个
	ROAM 算法,147-148 钻石池缓冲区, ROAM 算法,153-154
	钻石
	基本单元,149 创建 函数,151-153
	入队功能,158个合并功能,159个 父/子链接,150个优先级更新功 能,158个
坐标,纹理 (标准范围) ,40-41	ROAM算法,149-154拆分函数,158-159
CosineInterpolation 函数,185-186	拆分/合并优先级队列,156-160

菱形算法

—维解释,34-35
2D 解释,35-37 定义,33 缺点,
33

基于

雾距离,基于 189-190 个顶点,
191 个循环,程序纹理生成,
48-49

Direct3D,顶点缓冲区使用,82 个距离方程,

力缓冲,水渲染,172-174 公式,3D 距离,90-92

Roettger 的四叉树算法,112-113 基于距离的雾,
189-190

分形布朗运动 (FBM) 理论、云纹理,183-187 分形地形生
成定义,27 厄运过滤器,30-31 断层形成算法、

Duchaineau, Mark,ROAM 算法开发人员,128 种动
态光照贴图光照计算,68-69 种坡度光照计
算,69-71

27-33 中点位移算法,33-37

和

海拔

颜色表示,17-18 厄运过滤器,30-31 故
障形成算法,28-30

帧到帧相干性,ROAM 算法,136 频率,定义,184

截锥体剔除边界球,145-146

geomipmapping,98-101

Enqueue 函数,158 环境拓扑,
地形应用,5 厄运过滤器,高度图,30-31 误差
度量,ROAM 算法,134-136

ROAM 算法,144-148

Roettger 的四叉树算法,115-116,125

fTexBottom 函数,41 fTexLeft 函数,

41 fTexTop 函数,41

F

扇形中心,geomipmapping,94-96 断层形成算法
定义,27-28 高程,28-30 分形地形生成,27-33

功能

添加/删除,160

分配,160 亮度提取,

64

余弦插值,185-186

CreateParticle, 195-196 钻石

创建, 151-153 动态光照贴图, 68

FBM (分形布朗运动)理论,
183-187

入队,158 厄运

文件路径、CD 约定,8 个过滤
器

过滤器,31-33 释放,

侵蚀,30-31
对于,31-33
飞行模拟器,地形应用,5 个浮点运算,IEEE,139-141
个浮点变量,临时缓冲区,30

160 fTexBottom,41

fTexLeft,41 fTexTop,

41

GetBrightnessAtPoint, 64

212索引

功能 (续)

GetTrueHeightAtPoint, 58–59 高度检
索, 21 高度图操作, 21 iCurrentIteration,
29
iterations, 29
iMaxDelta, 29
iMinDelta, 29
初始化, 117
插值, 50 合并, 159 节点
细化, 118–119 节点渲
染, 111

噪音, 184 优先
更新, 158
传播粗糙度, 123–124 范围随机, 185–186

范围平滑随机, 185–186
RefineNode, 117, 124, 125 区域百
分比计算, 45–46
RenderChild, 139–142, 144
RenderFan, 93 渲
染, 119–121
渲染节点, 117
RenderVertex, 92–93, 117 重复纹
理, 51–52 缩放, 21

关机, 117 斜坡照明、
68–71 分割、 158–159 瓦片生
成、 47–48 瓦片管理、 47–48

没错, 21
更新, 90、 138、 161、 196–197

G

Game Developer Magazine, 78 游戏开
发, 3D 地形应用, 5–7

游戏编程宝石, 82

游戏

黑白, 5–6
马克思佩恩, 65 岁
地震, 2, 65
Starseige:部落, 5
足迹, 5–7
geomipmapping
3D 距离公式, 90–92
轴对齐边界框
(AABB), 99–100
CLOD 算法, 79–86 破解, 82–86 扇
中心, 94–96 截锥体剔除, 98–101
变形, 101–103 初始化, 88–89

细节层次 (LOD), 80–81
维护, 89–92 mipmapping
相似性, 79–80 邻居结构, 94–97 补丁渲染三
角形排列, 80–81

补丁结构变量, 87 弹出, 91–92, 101–
104
渲染函数, 92, 98
RenderFan 函数, 93 个渲染函
数, 92–98
RenderPatch 函数, 93–97
RenderVertex 函数, 92–93 屏幕像素确定
算法, 91–92

关闭, 89 个顶点缓
冲区, 81–82
变形, 减少弹出, 101–103
GetBrightnessAtPoint 函数, 64
GetTrueHeightAtPoint 函数, 58–59
全局分辨率标准方程,
Roettger 的四叉树算法, 112
gouraud 照明, 消除爆音,
103–104

显卡,顶点信息首选项,82

IEEE 浮点运算,ROAM 使用,139–141

灰度图像,高度图,17

iIterations 函数,29

GUI (图形用户界面),

iMaxDelta 函数,29

光盘,206

iMinDelta 函数,29

H

硬件照明,缺点,62–63 硬件多重纹理,细节贴图,

初始化函数,117 安装,

54–55 定义的基于高度的照明,58

CD-ROM,207 插值,粒子引擎,

199–200

大号

叶节点,Roettger 的四叉树算法,111

GetTrueHeightAtPoint 函数,58–59 光源着色,
59–62 光照值计算,59 缺点,60–62 视觉解释,59
高度图蛮力算法,24–27 数据缓冲区,22–23 描述,
16–18 高程颜色,17–18 个侵蚀滤波器,30–31 个
故障形成算法,27–33 个灰度图像,17 个加载,21–
23 个操作函数,21 个内存分配,22 个中点位移算法,
33–37 个

细节层次 (LOD)

裂缝修补,83–86

geomipmapping,80–81 寿

命,粒子属性,193 光源,着色,59–62 光

照动态光照贴图,68–71 gouraud,

103–104 硬件,62–63 基于高度,58–62

光照贴图,63–65 坡度,66–68 水渲

染,174 光照贴图亮度提取,64 定义,

63 动态,68–71 生成方法,63–64

像素,17 四

GetBrightnessAtPoint 函数,64 个

叉树矩阵,109–111

gouraud 光照,103–104 个加载,64 个链

RAW 格式,18 分辨

接

率依赖性,49–52

二叉三角形树节点,133–134 个菱形,150、

ROAM 算法,147–148 个保存例

154–155 循环程序纹理生成,48–49 而,51–52

程,23 个临时缓冲区,30 个卸载,

23–24 个无符号字符变量,16,18

我

iCurrentIteration 函数,29

214索引

米

- 主要演示,书中使用的约定, 8–10
- 质量,粒子属性,194矩阵,Roettger 的四叉树算法, 109–111
- McNally, Seamus (Treadmarks), 5–7
- McNally, Seamus, ROAM 算法改进,133 内存,高度图分配,22 合并函数,159 合并队列, ROAM 算法,132–133 Microsoft Visual C++,代码约定 在书中使用,8–10 中点位移算法 一维解释,34–35 2D 解释,35–37 定义,33 缺点,33 军事,地形应用,5 mipmapping, geomipmapping 相似之处, 79–80

ñ

- 邻居节点,二叉三角树节点链接,134 邻居结构,geomipmapping,94–97 噪音 柏林,184 粉色, 183–184 白色, 183 噪声函数,184 普通缓冲区,水渲染,172 NULL 指针,数据缓冲区,23–24

- OpenGL API,主要演示约定,8 OpenGL游戏编程 (阿斯特尔/霍金斯) ,54 户外场景,粒子引擎 申请,201–202

磷

- 粒子引擎空气阻力, 194 广告牌, 198–199 颜色, 194 CreateParticle 函数,195–196 当前位置,193 数据插值, 199–200 定义,192 发展史, 192 寿命, 193 质量, 194 户外场景应用, 201–202 户外使用, 192 关系, 192–193 大小, 194 结构元素,194–195 半透明,194 更新,196–197 速度, 194 粒子系统,已定义,192 补丁渲染 3D 距离公式,90–92 轴对齐边界框 (AABB) , 99–100 geomipmapping 三角形排列, 80–81 变形,101–103 if-else 语句 测试,84–85 邻居结构,94–97 更新函数,90 个变量,87 补丁,裂缝修补方法,83–86 柏林噪声,184 Perlin,Ken,Perlin Noise 开发人员,184 种粉红 噪声,已定义,183–184 像素亮度提取,64 种高度图,17 种等离子分形算法。见中点 位移算法

- 定义了弹出,91 变形,101-103 优先级队列,ROAM,定义了 156-160 程序纹理生成,43 个 for 循环,48-49 高度图分辨率依赖性,49-52 克服瓦片边界,51-52 区域系统,44-46 重复纹理,51-52 纹理数据,48-49 瓦片系统,46-48 PropagateRoughness 函数,123-124 传播,Roettger 的四叉树算法,123-124
- ## 问
- 尾巴 合并,132-133 拆分,132-133 拆分/合并优先级,156-160
- ## R
- 光能传递,光照贴图方法,65 个随机演示,书中使用的约定,8
- RangedSmoothRandom 函数,185-186 RAW 格式,高度图,18 个房地产演练,地形应用,5 实时最优适应网格 (漫游)算法 添加/删除功能,160 分配函数,160 个主干数据结构,149-156 个二叉三角树,129-132 个细节映射,147-148 个菱形创建函数,151-153 个菱形池缓冲区,153-154 个菱形,149-154 入队函数,158 错误度量简化,134-136 帧到帧一致性,136 释放函数,160 截锥体剔除,144-148 高度图,147-148 IEEE 浮点运算,139-141 合并函数,159 合并队列,132-133 节点改进,133-134 优先级更新函数,158 渲染函数,138-139,144,155-156,162-163 RenderChild 函数,139-142,144 关闭例程,142 拆分函数,158-159 拆分队列,132-133 拆分/合并优先级队列,156-160 仅拆分曲面细分,136 曲面细分基础节点,131-132 曲面细分级别,129-131 纹理映射,147-148 T-junction,131-132 Treadmarks,6 个三角形系统,160 更新功能,138,161 v0.25.0,137-144 v0.50.0,144-154 v0.75.0,154-160 v100 .0,161-163 Reeves, William T., 粒子引擎开发人员,192 RefineNode 函数,117,124,125 反射贴图,水渲染,171-172 区域系统,程序纹理生成,44-46 区域,已定义,44 RenderChild 函数,139-142,144 RenderFan 函数,93 RenderNode 函数,117 RenderVertex 函数,92-93,117

216索引

分辨率依赖性,高度图,49-52 ROAM。参见Real-Time Optimally Adapting Mesh algorithm Roettger, Stefan, Roettger's quadtree algorithm, 106 Roettger's quadtree algorithm 2D 数据结构, 106-108 自底向上方法, 115 个子节点, 110-111 CQUADTREE 类, 116-117 d2 计算, 114-在需要的地方添加了 115 个细节, 78 个距离方程, 112-113 个截锥体剔除, 115-116, 125 个全局分辨率标准方程, 112 个图示, 107-108 个初始化函数, 117 个初始化, 117 个叶节点, 111 个矩阵, 109-111 个节点细化函数, 118-119 节点细化, 118-119 节点渲染函数, 111 PropagateRoughness 函数, 123-124 传播, 123-124 RefineNode 函数, 117, 124, 125 渲染函数, 117 渲染, 117 渲染函数, 119-121 RenderNode 函数, 117 RenderVertex 函数, 117 根四边节点渲染, 110-111 关闭, 117 嵌入, 110 自顶向下方法, 112 更新, 117 中上顶点节点计算, 123-124

小号

屏幕像素确定算法,避免弹出,91-92

Shankel, Jason, FIR 滤波器建议, 31

关机功能, 117

大小, 粒子属性, 194 天空盒, 特效, 175-178 天穹云纹理, 182-187

CosineInterpolation 函数, 185-186 定义, 178

分形布朗运动 (FBM) 理论, 183-187 代讨论, 179-182

柏林噪声, 184

范围随机函数, 185-186

RangedSmoothRandom 函数, 185-186

渲染, 182 斜坡照

明 45 度增量, 67 自定义功能, 68

定义, 66

动态光照贴图, 68-71 个视觉示例, 67 个特效相机-地形碰撞检测,

187-189

基于距离的雾, 189-190 粒子引擎, 192-202

柏林噪声, 184

sky-box, 175-178

sky-dome, 178-187

vertex-based 雾, 191

water rendering, 166-174

Special Effects Game Programming with DirectX 8.0 (McCuskey),

54 Split function, 158-159 split

queue, ROAM algorithm, 132-133 拆分/合并优先级队列, ROAM 算法, 156-160

根四节点, Roettger 的四叉树
算法, 110-111

仅拆分曲面细分,ROAM 算法,136 条语句,if-else,

84-85

结构

BinTriNodes,134-135

光盘内容,207

STRN_TEXTURE_REGIONS,44

STRN_TEXTURE_TILES,47 系统

要求,CD-ROM,206

吨

临时缓冲区,创建,30 个地形,已定义,

4 个地形编程

3D 游戏开发,5-7 个应用程序,4-5

个分形地形生成,27-37 个高度图,

16-18 个镶嵌

基本节点,131-132 二叉

三角树,129-132 d2 计算,114-115

合并队列,132-133 四叉树矩阵,

110 拆分队列,132-133 仅拆分,

136 纹理坐标,标准范围,40-41 纹

理数据,程序纹理生成,48-49

瓷砖

克服边界,51-52 区域系统,44-46 纹

理贴图,42-43 平铺系统,46-48

T-junction, tessellating binary triangle tree

base nodes, 131-132 translucency, 粒子

属性, 194 旅行规划（虚拟旅游）, 地形应用, 5

纹理贴图细节贴

图,52-54 高度图分辨

率依赖性,49-51 程序纹理生成,43-52

真正功能,21

Turner, Bryan, ROAM 教程,136

紫外线

更新函数,90,138,161,196-197 个变量

浮点数,30 个补丁结

构,87 个

RGB 值提取,48-49 ucShade,59-

60 uiS,140-141 unsigned char,

16,18 vecLightColor,60 速度,粒

子属性,194 顶点缓冲区

ROAM 算法,147-148 简单应用,

40-43 纹理存在线,44-45 瓦片,

42-43 顶点渲染,41 水渲染,168-

169 纹理存在线,纹理映射,44-45

瓦片系统,程序纹理生成,46-48

基于顶点的雾,191 个顶

点,渲染,41 个顶点,裂缝修

补方法,83-86

查看片断

边界球测试,145-146 geomipmapping,

98-101 虚拟旅游（旅游规划）,地形应

用,5

218索引

在

水渲染 alpha 混合,

169 力缓冲,172-174

照明,174 普通缓

冲区,172

四边形设置,167-168 反射贴

图,171-172 纹理添加,168-169

纹理动画,170

顶点缓冲区,172

天气可视化,地形应用,5

网站

算法白皮书, 79 作者的 ROAM 算法实

现, 144 Flipcode, 203 GameDev.net,

7, 203

肯·佩林,184

马克·莫利的截锥体剔除

OpenGL,99

光能传递技术,65

Virtual Terrain Project, 4, 203 白噪声,

定义, 183

X-Z

X轴

for 循环,48-49 高度

图,16 插值计算,50-51

Y 轴,高度图,16

Z轴

for 循环,48-49 个高度

图,16 个插值计算,51

个

GAME DEVELOPMENT.

IT'S SERIOUS BUSINESS.

“毫无疑问,游戏编程是世界上最具智力挑战性的计算机科学领域。但是,如果说我们是“认真”的人,那就是自欺欺人了!编写(和阅读)游戏编程书籍对于作者和读者来说都应该是一次激动人心的冒险。”

安德烈·拉莫特,
丛书编辑



Premier Press
A Division of Course Technology
www.premierpressbooks.com



Gamedev.net

The most comprehensive game development resource

- The latest news in game development
- The most active forums and chatrooms anywhere, with insights and tips from experienced game developers
- Links to thousands of additional game development resources
- Thorough book and product reviews
- Over 1000 game development articles!

Game design

Graphics

DirectX

OpenGL

AI

Art

Music

Physics

Source Code

Sound

Assembly

And More!

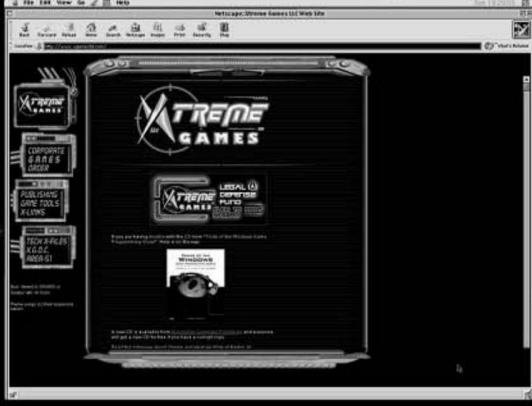


Gamedev.net

OpenGL is a registered trademark of Silicon Graphics, Inc.

Microsoft and DirectX are registered trademarks of Microsoft Corp. in the United States and/or other countries.

带上你的
游戏到
极限！



Xtreme Games LLC的成立旨在帮助世界各地的小型游戏开发商在商业市场上创建和发布他们的游戏。Xtreme Games帮助年轻的开发者打入游戏编程领域,让他们远离复杂的法律和商业问题。Xtreme Games拥有数百个开发者

在世界各地运营。如果您有兴趣成为其中一员,请访问我们的
www.xgames3d.com

www.xgames3d.com



许可协议/有限保修通知

打开本书中的密封圆盘容器,即表示您同意以下条款和条件。如果在阅读以下许可协议和有限保修通知后,您不能同意所规定的条款和条件,请将未使用的图书连同未开封的光盘退回您购买它的地方以获得退款。

执照:

随附软件的版权归软件光盘上注明的版权所有者所有。您被许可将软件复制到单台计算机上供单个用户使用并复制到备份光盘上。除非获得版权所有者的书面许可,否则您不得复制、制作副本或分发副本或出租或出租全部或部分软件。您只能将随附的光盘与本许可一起转让,并且只有在您销毁软件的所有其他副本且受让人同意许可条款的情况下才能转让。您不得对软件进行反编译、反向汇编或反向工程。

有限保修通知: Premier Press, Inc.

保证随附的光盘在最终用户购买图书/光盘组合后的六十 (60) 天内不会出现材料和工艺方面的物理缺陷。在 60 天的有限保修期内,Premier Press 将在退回有缺陷的光盘时提供更换光盘。

有限责任:违反本有限保

证的唯一补救措施应完全包括更换有缺陷的光盘。在任何情况下,Premier Press 或作者均不对任何

其他损害,包括数据丢失或损坏、更改

在硬件或操作的功能特性中

系统、与其他软件的有害交互,或可能出现的任何其他特殊、附带或后果性损害,即使 PREMIER 和/或作者之前已被告知存此类损害的可能性。

免责声明: Premier 和作者明确否

认任何和所有其他明示或暗示的保证,包括对适销性、特定任务或目的的适用性或免于错误的保证。某些州不允许排除

附带或间接损害的默示保证或限制,因此这些限制可能不适用于您。

其他:

本协议受印第安纳州法律管辖,不考虑法律选择原则。国际货物销售联合合同公约被明确否认。本协议构成您与 Premier Press 之间关于软件使用的完整协议。