罗笑南　潘志庚　纪庆革　高成英　主编

数字娱乐与艺术进展——第三届全国数字娱乐与艺术暨数字家庭交互应用技术与设计学术研讨会论文集

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

数字娱乐与艺术进展——

第三届全国数字娱乐与艺术暨数字家庭交互应用技术与设计学术研讨会论文集

罗笑南　潘志庚　纪庆革　高成英　主　编

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

第三届全国数字娱乐与艺术暨数字家庭交互应用技术与设计学术研讨会（DEA'2008）是由中国系统仿真学会数字娱乐仿真专业委员会、中国图像图形学会虚拟现实专业委员会和SIGCHI中国分部联合主办，前两届会议分别在成都和南京举行，本届会议由中山大学数字家庭教育部重点实验室承办。大会于2008年11月8日～10日于广州举行，共收到来稿论文120篇，经过专家评审，录用了其中40篇，内容涉及数字娱乐、虚拟现实、真实感图形图像技术、人机交互、CAD、视频、图像和数字信息安全等领域，在不同程度上反映了中国数字娱乐的发展。

本书的内容无论是对于从事数字娱乐艺术的艺术创作者，还是从事图形学相关研究的专家、学者、研究生都有较大的参考价值。

图书在版编目（CIP）数据

数字娱乐与艺术进展：第三届全国数字娱乐与艺术暨数字家庭交互应用技术与设计学术研讨会论文集／罗笑南等主编. 北京：电子工业出版社，2009.12

ISBN 978-7-121-09411-8

Ⅰ. 数… Ⅱ.罗… Ⅲ.①数字技术－应用－文娱活动－服务业－中国－文集②数字技术－应用－艺术－中国－文集 Ⅳ.F719.5-39 J-39

中国版本图书馆CIP数据核字（2009）第142435号

责任编辑：史 涛

印 刷：北京季蜂印刷有限公司

装 订：北京季蜂印刷有限公司

出版发行：电子工业出版社

北京市海淀区万寿路173信箱 邮编 100036

开 本：787×1 092 1/16 印张：19.5 字数：499千字

印 次：2009年12月第1次印刷

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

第三届全国数字娱乐与艺术暨数字家庭交互应用技术与设计学术研讨会

DEA'2008

主办单位：

中国系统仿真学会数字娱乐仿真专业委员会

中国图像图形学会虚拟现实专业委员会

SIGCHI 中国分部

承办单位：

中山大学数字家庭教育部重点实验室

第三届全国数字娱乐与艺术暨数字家庭交互应用技术与设计学术研讨会

DEA'2008

名誉主席：李伯虎 中国系统仿真学会、中国工程院院士

大会主席：罗笑南 （中山大学）
王阳生 （中科院自动化研究所）
戴国忠 （中科院软件所）

程序委员会主席：潘志庚 （浙江大学）
王涌天 （北京理工大学）
陆哲明 （中山大学）

程序委员会成员（中文姓名按拼音排序）：

陈 洪	陈东义	戴国忠	高 文	韩国强	侯格贤
纪庆革	金小刚	康 波	李 毅	李 艺	林 超
林小拉	刘直芳	陆哲明	罗钟铉	罗笑南	潘志庚
檀结庆	王建民	王若梅	王阳生	王涌天	徐心和
杨旭波	叶 凤	尹宝才	由 芳	战荫伟	张祥和
张 军	Ruth Aylett		Ji-hong Jeung		Guojun Liao
Xiaoyan Liu	李笑迎	邹湘军	彭 强	朱学芳	刘惠义
赵向军	王 辉	谭群钊	王鸿冀	叶 展	谢仕义

组织委员会主席：林小拉

组织委员会秘书长：纪庆革（秘书长） 王建民（副秘书长）
侯格贤（副秘书长）

组织委员会委员：陈华鸿 高成英 侯格贤 纪庆革 毛志红 王建民 周 凡

前　言

第三届全国数字娱乐与艺术暨数字家庭交互应用技术与设计学术研讨会（DEA'2008）是由中国系统仿真学会数字娱乐仿真专业委员会、中国图像图形学会虚拟现实专业委员会和 SIGCHI 中国分部联合主办，前两届会议分别在成都和南京举行，本届会议由中山大学数字家庭教育部重点实验室承办。大会于 2008 年 11 月 8 日～10 日举行，共收到来稿论文 120 篇，经过专家评审，录用了其中 40 篇。

本书内容包括人机交互、娱乐仿真技术、数字家庭、交互数字娱乐及增强现实、数字电视设计、游戏中的人工智能与人工生命、数字艺术、真实感图形图像技术、广告印刷/工业设计/服装设计的 CAD 应用系统、虚拟人物与 Agent、虚拟现实、CAD/CG 等众多领域，在不同程度上反映了中国数字娱乐的发展。本书的内容无论是对于从事数字娱乐艺术的艺术创作者，还是从事图形学相关研究的专家、学者、研究生都有较大的参考价值。

非常感谢石晓红老师、刘成明博士后等对此次大会筹备工作的大力支持。

感谢中山大学信科院硕士生彭伟、牟宁、高水波、冯华、张燕平、李嘉豪、于鸿磊、徐攀雄、蔡海滨、熊晨辉、冯建伟和本科生杨铭等同学，他们在网站设计、维护、会务组织、论文编辑及其他会务工作中表现出色，付出了辛勤劳动。

编辑工作不当之处，敬请谅解。在此再次感谢所有对本次大会（DEA' 2008）做出贡献的人们！

罗笑南　潘志庚　纪庆革　高成英

2008 年 11 月

目　录

第一部分　数字娱乐、虚拟现实、真实感图形图像技术

第二部分　人机交互、CAD

第三部分 视频、图像、数字信息安全

第一部分

数字娱乐、虚拟现实、真实感图形图像技术

棋盘类游戏中的栅格地形渲染

张嘉华　梁　成　李桂清

（华南理工大学计算机科学与工程学院　广东　广州　510640）

摘　要：本文提出了适合战棋类游戏的三维栅格地形渲染策略，把 Catmull－Clark 细分曲面思想应用到规则栅格地形，探讨了栅格内顶点和像素的各种属性的计算方法，实现了栅格之间的平滑过度，介绍了在 GPU 中的实现细节，采用了纹理样式集，几何实例，顶点程序段纹理访问等技术。本文还给出了 Vertex Shader 和 Pixel Shader 的伪代码，有一定 3D 图形编程能力的游戏开发人员能够快速再现文本工作。

关键词：战棋类游戏；栅格地形；图形处理器；几何实例

Rendering Grid Terrain for SLG Games

Zhang Jiahua　Liang Cheng　Li Guiqing

(College of Computer Science and Engineering, South China University Of Technology , Guangzhou Guangdong, 510640, China)

Abstract: This paper proposes a 3D grid terrain rendering approach for SLG games, applying the idea of Catmull－Clark Subdivision Surface to regular terrain grids, Methods for calculating various attributes of vertex and pixel in the grid is introduced. This paper also introduces the detail implement on GPU, using technologies such as style texture collection, geometry instancing, vertex texture fetch. This paper gives the pseudocodes of vertex shader and pixel shader. it is helpful for game programmer to implement our work in a short time.

Keywords: SLG; Grid tTerrain; GPU; Geometry Instancing

1　引言

战棋类游戏（SLG）主要在栅格地形上进行，玩家根据行动顺序指定自方角色沿着栅格行动，进行战斗或冒险。这一类游戏有着庞大的代表性游戏：《超时空英雄传说》系列、《火焰之纹章》、《神奇传说》系列、《天使帝国》系列等。以往都是在二维平面上展现游戏棋盘栅格，随着三维图形技术，特别是现代 GPU 的迅速发展，已经有很多有代表性的三维 SLG 游戏，例如《三国志 11》、《信长野望之革新》、《英雄无敌 5》等。因此，三维栅格地形的渲染成为 SLG 游戏图形模块重要的一部分。

如图 1 所示，游戏场景设计人员往往在地图这些栅格上进行编辑，设置栅格的颜色，纹理样式等来表示这个栅格是雪地、草地还是水体，对于三维 SLG 游戏还要设置这个栅格的地形高度。

随着游戏的发展，每个栅格若采用单调的颜色来表示往往不能满足需求，因此希望对于场景设计人员来说仍然能够以栅格为单位设置属性，而且希望栅格与栅格之间能够表现出渐变过渡。如图 2，从左到右依次经历水体、沙地、泥地、草地、沼泽地的变化。那么，如何能够既让地形编辑人员能够方便地对每个地形栅格设置属性，同时又能在栅格之间表现出平滑的过度，就是本文研究的重点。

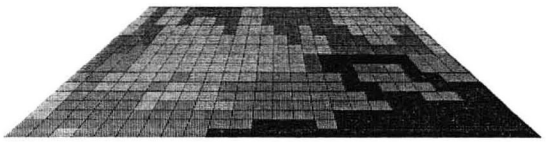


图 1　SLG 游戏的地图栅格

图 2　带有渐变和过度的游戏地图栅格

本文的主要贡献包括：①介绍了一种适合 SLG 类游戏栅格地形绘制的方法；②介绍了栅格内顶点和像素的属性位置计算方法，实现了栅格之间的平滑过渡；③介绍了 GPU 实现细节，采用了纹理集，Geometry Instancing，Vertex Texture Fetch 等技术，并给出了 Vertex Shader 和 Pixel Shader 的伪代码。有一定 3D 图形编程能力的游戏开发人员能够快速再现文本工作。

接下来的章节将如下组织：首先介绍对栅格地形绘制有影响的相关工作；其次介绍本策略栅格地形绘制的基本框架；再其次介绍了栅格内顶点和像素的属性位置计算方法，并且为后面的 GPU 实现提供了理论基础；最后给出了本文方法的运行效果。

2　相关工作

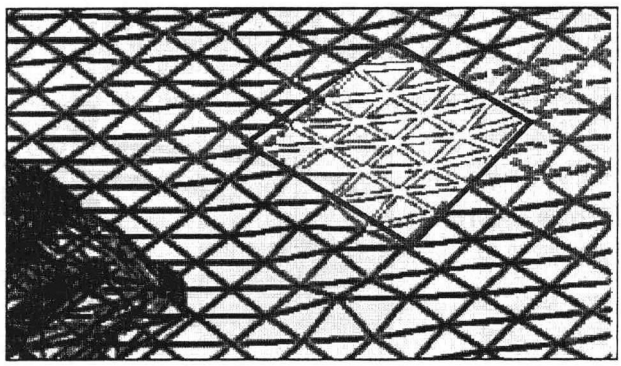
地形的绘制已经有非常多成熟的方案，也有许多适合图形硬件的方案，大致可以分为基于二叉树、四叉树等的 LOD 方案，适合硬件加速的分块 LOD 方案，视点依赖的针对 GPU 的几何裁剪图方案等。

[Duchaineau et al, 1997]提出采用 ROAM 二元自适应合并分割的方法，通过实时合并和分割三角形来调整地形精度，需要在 CPU 上不断进行调整。该方法具有灵活的视点依赖的误差评价，用户能够指定三角形数量，具有帧一致性（frame-to-frame coherence）等优点，但该方法并不符合现代 GPU Large Batch 和少 DIP Call 的原则。[Röttger et al, 1998]提出对于地形高程绘制采用连续 LOD 模型来处理，使用四叉树来描述地形高程，对内存需求很少，支持 geomorphing，并且能够实现快速处理地形高程的分页（paging of large elevation grids）。

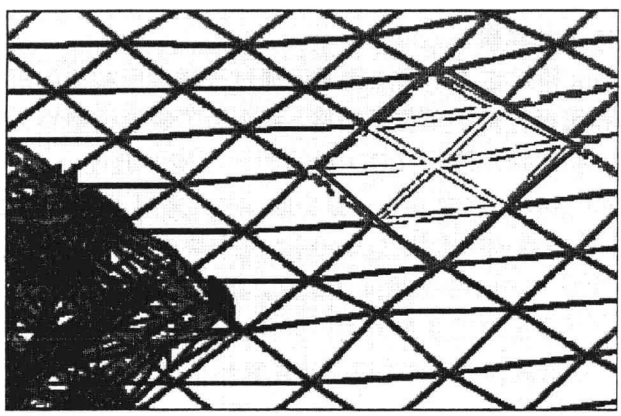
随着 2002 年后 GPU 的迅速发展，不再需要在 CPU 上进行大量严格简化，而需要在 CPU 简化和 GPU 批量处理之间取得一个较好的平衡点。[Ulrich, 2002]提出了分块 LOD 方案。每一个块（chunk）是一个规则格网集，在格网边缘采用垂直裙子方法来填补块与块之间的裂缝。由于每块是一个离散化的整体，有利于 GPU 批量绘制，该方法有低 CPU overhead、高渲染吞吐量、能够在 aggregate 内使用三角网、纹理 LOD 与地形几何 LOD 相适应、out-of-core 存储、有效平滑的顶点渐变（morphing）、无顶点跃动（poping）和在视点移动时低 CPU 负荷

等优点。但也具有下面这些缺点：数据集必须是静态的，比传统的动态 LOD 方案需要使用更多的三角形、块内无法进行遮挡剔除。[Larsen, 2003]提出采用硬件优化的 LOD 地形方案，与 Chunk LOD 相似基于规则格网四叉树，不同的是对于每个块内部，再内建自适应四叉树，使用 Vertex Shader 在每个顶点上处理 geomorphing。[Ulrich, 2002]提出的分块 LOD 方法对于栅格地形来说是适用的，因为可以把地形上若干栅格组织成一个块，以块为单位进行视锥剔除和绘制，能够很好地利用 GPU，[Larsen，2003]提出的方法则为我们在 Vertex Shader 计算顶点属性和位置提供了启发。

《信长野望之革新》与《三国志 11》的地形引擎和栅格划分类似，都是对地形划分为一个个栅格，如图 1，对于棋盘地形中的每个栅格，都进行 LOD 处理。这种 LOD 类似于[Ulrich, 2002]的方法，从测试中只观察出两级 LOD。对于图 3（a），白色区域选定的栅格由 4×4×2 个规则三角片构成，当摄像机镜头拉离地表时，LOD 级别发生了改变，选定的白色区域的栅格变为图 3（b）中的 2×2×2 个规则三角片构成。



(a)



(b)

图 3　三国志 11 在 GameAssassin1.4 下的线框渲染，高亮区域为鼠标选定的某个栅格

3　基本框架

整个地形都是按栅格进行划分的。图 4 是一个栅格（grid）的定义，栅格的纹理样式和

颜色都定义在中心采样点上，输入 256×256 的原始地形，就有 256×256 个栅格。每个栅格由四个子区域（sub rect）组成，每个子区域由 4 个顶点构成。读者肯定会问一个问题，为何不直接对每个栅格用一个顶点表示其中心点呢？这是因为不同栅格的各种属性之间需要过渡，并且边界上的顶点属性并不是都刚好满足周围 4 个采样点的双线性插值关系，需要在栅格边界上放置经过计算的顶点以保证属性过渡正确，并且使栅格内非边界区域能展现该栅格设置的采样点属性。

因此，每个子区域 2×2 个顶点，共 2×2×4=16 个顶点。对于一个 256×256 原始采样点的图来说，那么就需要 256×256×16 个顶点。那么读者又会问，为何不是一个栅格只要周围 8 个边界上的顶点和一个中心顶点呢？这是由于每个子区域都是由它周围 4 个采样点按照一定比例混合过渡的。如图 2，8 号顶点和 2 号顶点则被看做在不同的子区域，对它有权重影响的周围 4 个采样点是不一样的，因此两个顶点的 4 层纹理坐标也有差别。

地图设计编辑人员往往希望只需要对每个栅格独立设置各种属性，包括纹理样式和颜色等，就能产生平滑过渡的地形，这和细分曲面很类似，往往希望只修改低分辨率的控制网格但产生高分辨率光滑的网格。因此，对某个栅格(i,j)设置属性就等价于为中心采样点设置属性。栅格中心采样点的某种属性定义为 $a(i,j)$，栅格中每个顶点也拥有自己的属性，如图 4 中构成该栅格的 16 个顶点的对应属性为 $\{a_k(i,j)|k=0,1,..,15\}$。这里需要说明的是，v_3，v_6，v_9，v_{12} 四个顶点从二维投影来看与采样点是重合的，但是从三维空间上来看，最终计算得到的位置并不一定与采样点重合的，这取决于计算模板，对于 Catmull－Clark 模式这 4 个顶点将逼近通过采样点所构造的极限光滑曲面。

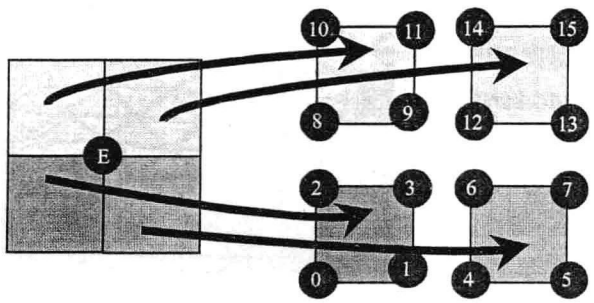


图 4　1 个栅格的定义

考虑到用户编辑和选择的最小单位是栅格，那么为了表现该栅格，最小需要 16 个顶点来表现一个栅格。因此，在对地形进行 LOD 处理时，不能够通过合并栅格来减少顶点数量，至少也要维持每个栅格 4×4=16 个顶点。有鉴于此，本文采用了[Ulrich, 2002]的方案来分块进行处理，每 4×4=16 个栅格组成一个块（chunk），每个块内 LOD 时最低分辨率约束不能低于（4×4）×（4×4）=256 个顶点。LOD 采用 4 个级别，最高级别时每个块内有 128×128=16384 个顶点。由于栅格的子区域四层纹理是不同的，因此，不能对栅格与栅格边界上的顶点进行合并减少顶点。在每帧，视锥剔除和 LOD 级别计算等都是以块为单位进行。

对于地形范围比较少的场景，也就是采样点少于等于 256×256 的场景，可以预先构建好整个地图所有的块，并生成每个块的所有 LOD 级别，但是这样存在较多空间冗余，因为每帧绘制中，大部分的块都采用较低的 LOD 级别，而采用较高的级别只有 1 至 2 个视点正在关注的感兴趣的块。如果，每个块都生产好各个 LOD 级别，一个块所有 LOD 级别共需要 256+1024+4096+16384=21760 个顶点，整个 256×256 个栅格的场景共有 64×64=4096 个块，那么整个场景生成时就有 21760×4096=89128960 个顶点，而实际每帧绘制的地形也就约 2×16384+6×4096+16×1024+64×256=90112 个顶点（2，6，16，64 为平均每周视锥剔除后场景中各个 LOD 级别块的数量），也就是绘制的顶点约为总顶点数量的 0.1%。为了解决这个问题，

已经有许多 Out-Of-Core 的外存算法以及内存分页映射的加载调度算法，这些算法对于海量数据（例如 TB 级的数字地球的金字塔影像数据库）来说是合适的，但是我们整个场景只有 256×256 个栅格，也就是 65536 个采样点，大部分高分辨率 LOD 级别的顶点数据都是插值得到的，并不是真的存在那么多高分辨率采样点，之所以存在那么多顶点，主要一个原因就是对应一个采样点的栅格就产生了至少 16 个顶点。

由于整个场景的栅格一般只有 256×256 数量，没有必要在运行时进行内外存数据交换，但是我们又希望当镜头靠近时能插值出高分辨率的顶点。因此，是否有一种无需生成具体顶点，能够延迟具体顶点位置和属性计算到绘制时再在 GPU 处理的方法呢？

Geometry Instancing 技术和 Vertex Texture Fetch 技术使得这样的想法成为了现实。在 CPU 中，只要为其中一个块生成好各个 LOD 级别的顶点缓冲，这里记这些顶点缓冲为 VB_β，β 为 LOD 级别，取 0～3。顶点缓冲 VB_β 中各个顶点的位置只需要填上相对于块内的位置，绘制时在 Vertex Shader 再计算具体的位置。有了这样的基础后，绘制时对于每个块根据自己当前帧的 LOD 级别选择对应的 VB_β 来绘制。如果采用 Geometry Instancing 方式绘制，为同一个 LOD 的各个块设置一个实例数组放到实例缓冲，通过 SetStreamSourceFreq 设置后绘制；对于不支持 Geometry Instancing 的 GPU，可以对于同一个 LOD 级别的各个块分别调用 DIP call。这使得同处于 β 这个 LOD 级别的各个块都可以共享一个顶点缓冲 VB_β。这大幅度减少了顶点数量，在 CPU 分配顶点缓冲时只需要 21760 个顶点。

为了能够在绘制时再计算出具体的顶点位置和属性，因此需要用到 Shader Model 3.0 的 Vertex Texture Fetch 技术。首先，我们把场景地图中的 256×256 个采样点的位置、颜色和选用的纹理样式分别存放到 3 张纹理 *TP*、*TC* 和 *TS*。*TP* 中的一个像素 $TP(i,j)$对应采样点的位置，记为 $p(i,j)$，*TC* 中一个像素 $TC(i,j)$对应采样点的颜色，记为 $c(i,j)$，*TS* 中一个像素 $TS(i,j)$对应采样点选用的纹理样式 $t(i,j)$。在 GPU 的 Vertex Shader 根据输入的每个待插值顶点在块内的编号得到顶点在栅格内的编号 k，通过 tex2DLOD 指令从这 3 张纹理读出临近 4 个采样点的数据，加权得到这个插值顶点的数据。具体顶点的属性和位置计算在第 3 节介绍。

4　属性和位置的插值计算

第 3 节介绍了栅格地形绘制的基本思路，那么如何计算栅格内的 16 个顶点和栅格覆盖的各个像素的属性和位置呢？下面 4.1 节将介绍顶点除位置外各种属性包括纹理、颜色的计算，4.2 节进一步介绍像素属性的计算，4.3 节介绍顶点的位置计算。

4.1　顶点属性计算

为了实现栅格间的各种属性（纹理，颜色等）的融合，先来考察最为简单的双线性混合。下面推导栅格上每个顶点的属性权重比例。对于浅黄色子区域（8-9-10-11），参见图 5，实质上是采样点 *A-B-D-E* 所构成的区域的右下角，因此顶点 9 的某个属性如下计算：$a_{10}=w_{10}(a_A, a_B, a_D, a_E)^T$，$w_i$ 为四维向量，分别代表影响该子区域的四种类型纹理的权重，其分量顺序对应每个子区域的纹理样式编码，均为左下，左上，右下，右上，例如 $w_{10}=(0.25,0.25,0.25,0.25)$，类似可以得到下面关系：

$$
\begin{cases}
a_0 = (0.25 \quad 0.25 \quad 0.25 \quad 0.25) \bullet A_{GDHE} \\
a_1 = (0.0 \quad 0.0 \quad 0.5 \quad 0.5) \bullet A_{GDHE} \\
a_2 = (0.0 \quad 0.5 \quad 0.0 \quad 0.5) \bullet A_{GDHE} \\
a_3 = (0.0 \quad 0.0 \quad 0.0 \quad 1.0) \bullet A_{GDHE}
\end{cases}
$$

$$
A_{GDHE} = (a_G \quad a_D \quad a_H \quad a_E)^T
$$

$$
\begin{cases}
a_4 = (0.5 \quad 0.5 \quad 0.0 \quad 0.0) \bullet A_{HEIF} \\
a_5 = (0.25 \quad 0.25 \quad 0.25 \quad 0.25) \bullet A_{HEIF} \\
a_6 = (0.0 \quad 1.0 \quad 0.0 \quad 0.0) \bullet A_{HEIF} \\
a_7 = (0.0 \quad 0.5 \quad 0.0 \quad 0.5) \bullet A_{HEIF}
\end{cases}
$$

$$
A_{HEIF} = (a_H \quad a_E \quad a_I \quad a_F)^T
$$

$$
\begin{cases}
a_8 = (0.5 \quad 0.0 \quad 0.5 \quad 0.0) \bullet A_{DAEB} \\
a_9 = (0.0 \quad 0.0 \quad 1.0 \quad 0.0) \bullet A_{DAEB} \\
a_{10} = (0.25 \quad 0.25 \quad 0.25 \quad 0.25) \bullet A_{DAEB} \\
a_{11} = (0.0 \quad 0.0 \quad 0.5 \quad 0.5) \bullet A_{DAEB}
\end{cases}
$$

$$
A_{DAEB} = (a_D \quad a_A \quad a_E \quad a_B)^T
$$

$$
\begin{cases}
a_{12} = (0.5 \quad 0.0 \quad 0.5 \quad 0.0) \bullet A_{EBCF} \\
a_{13} = (0.5 \quad 0.0 \quad 0.0 \quad 0.5) \bullet A_{EBCF} \\
a_{14} = (0.5 \quad 0.5 \quad 0.0 \quad 0.0) \bullet A_{EBCF} \\
a_{15} = (0.25 \quad 0.25 \quad 0.25 \quad 0.25) \bullet A_{EBCF}
\end{cases}
\tag{1}
$$

$$
A_{EBCF} = (a_E \quad a_B \quad a_C \quad a_F)^T
$$

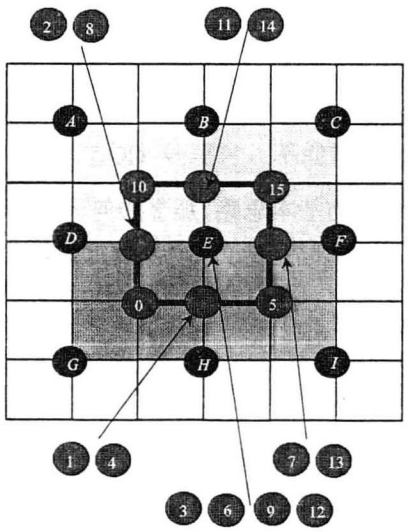


图 5　某个栅格 E 与其邻接 8 个栅格示意图，棕色边界为栅格 E 的范围。

4.2　像素属性计算

不但要根据采样点得到顶点颜色，更进一步需要根据采样点得到栅格内某个像素的颜色。

虽然现代的GPU都已经能够在硬件实现双线性插值，但是为了后文讨论方便，下面还是先对每个像素的属性计算先作进一步推导。

假设栅格坐标如图6，栅格内顶点像素范围取值从-0.5至0.5，取 $u1$=floor(u)，$u2$=ceil(u)，$v1$=floor(v)，$v2$=ceil(v)，因此 $u1$，$u2$，$v1$，$v2\in[-1,1]$，可以取到相邻栅格的采样点。那么对于栅格内任意一个像素，影响其的四个栅格（采样点）分别为（i+$u1$，j+$v1$），（i+$u2$，j+$v1$），（i+$u1$，j+$v2$），（i+$u2$，j+$v2$），对于图4，假设像素处于右下角子区域，那么 $u1$=0，$u2$=1，$v1$=−1，$v2$=0，影响它的四个栅格，也就是提供它融合权重的四个采样点分别为（i+0，j−1），（i+1，j−1），（i+0，j+0），（i+1，j+0）。接着计算混合比例 u'=fmod(u，1.0)，v'=fmod(v，1.0)，即对1.0求浮点模，那么栅格内任意点(u，v)的属性可以通过公式（2）双线性插值计算得到：

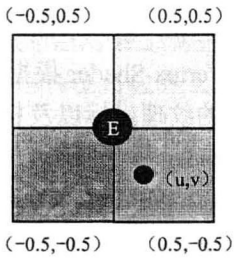


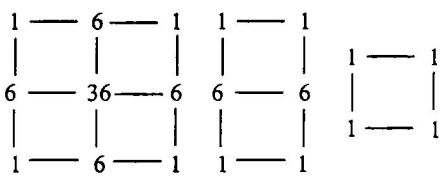
图6　逐像素纹理混合

$$
\begin{aligned}
a(u,v1) &= (1-u')a(i+u1,j+v1)+u'a(i+u2,j+v1)\\
a(u,v2) &= (1-u')a(i+u1,j+v2)+u'a(i+u2,j+v2)\\
a(u,v) &= (1-v')\times a(u,v1)+v'\times a(u,v2)
\end{aligned}
\tag{2}
$$

4.3　顶点位置计算

考虑到每个栅格的顶点位置定义在栅格的中心采样点上，为了使每个栅格周围的16个顶点位置能够更加平滑，因此可以考虑采用细分曲面的Catmull−Clark细分模式来计算16个顶位置。

根据图7中Catmull−Clark细分模式的三种模板，从图5可以看出，0，5，10，15号顶点属于 F 顶点，1，2，4，7，8，11，13，14号顶点属于 E 顶点，3，6，9，12号顶点属于 V 顶点，因此有下面的式子：



（a）V顶点模板　（b）E顶点模板　（c）F顶点模板

图7　Catmull−Clark细分模板

$$
\begin{aligned}
P_0 &= (P_D+P_E+P_G+P_H)/4\\
P_5 &= (P_E+P_F+P_H+P_I)/4\\
P_{10} &= (P_A+P_B+P_D+P_E)/4\\
P_{15} &= (P_B+P_C+P_E+P_F)/4\\
P_1 &= P_4 = (6P_E+6P_H+P_D+P_F+P_G+P_I)/16\\
P_2 &= P_8 = (6P_E+6P_D+P_A+P_B+P_G+P_H)/16\\
P_7 &= P_{13} = (6P_E+6P_F+P_B+P_C+P_H+P_I)/16\\
P_{11} &= P_{14} = (6P_B+6P_E+P_A+P_D+P_C+P_F)/16\\
P_3 &= P_6 = P_9 = P_{12} = (36P_E+6P_B+6P_D+\\
&\quad 6P_F+6P_H+P_A+P_C+P_G+P_I)/64
\end{aligned}
\tag{3}
$$