文章编号：2095-4980(2015)02-0297-05

PN 三角形在基于 GPU 曲面细分中的应用

陈 驰[a,b]，吴志红[a,b*]

(四川大学 a. 视觉合成图形图像技术国防重点学科实验室；b. 计算机学院，四川 成都 610065)

摘 要：为了增强模型渲染的真实感，本文使用曲面点–法线(PN)三角形或四边形(原理与三角形类似)来细化和调整输入模型中的三角形或四边形面片，从而实现棱角的柔和化。在 GPU(图形处理器)管线和图形库(如 DirectX 10 和 11)支持细分的前提下，该技术能适用于现有的渲染管道结构，并且对建模工具也没有特别的要求，仅基于顶点位置和法线信息便可渲染出圆润的轮廓，同时使用自适应细节层次(LoD)技术，提高了渲染的效率，再加上法线贴图和置换贴图的应用，可渲染出外形贴近真实且更具质感的效果，因而具有一定的实用价值。

关键词：曲面点–法线(PN)三角形；细分；GPU 运算；DirectX 11 图形库；渲染

中图分类号：TN911.73；TP391.9 **文献标识码：**A **doi：**10.11805/TKYDA201502.0297

Application of PN triangles in GPU based tessellation

CHEN Chi[a,b]，WU Zhihong[a,b*]

(a. National Key Laboratory of Fundamental Science on Synthetic Vision;
b. Computer Science Department, Sichuan University, Chengdu Sichuan 610065, China)

Abstract: Curved Point–Normal(PN) triangles or the similar PN quads are both efficient ways to enhance realism in 3D rendering through tessellating every input triangle or quad into smaller triangles based on tessellation factor and soften triangles' shared edges by using Bezier techniques. With the support of tessellation in modern GPU(Graphic Processing Unit) architecture and graphics library (e.g. DirectX 10 and 11), the PN triangles or quads has neither special requirement on rendering pipeline nor modeling tools, and it can produce soft silhouette and efficient rendering result by using vertex position, normal information, adaptable Load of Detail(LoD), and possible Normal Map or Displacement Map.

Key words: PN triangles; tessellation; GPU calculation; DirectX 11; rendering

建模软件输出的模型是三角形面片的集合，渲染效果的好坏受模型质量高低的影响，如果模型外表粗糙，在不进行曲面细分的情况下，渲染效果会变得过于突兀，从而显得不真实。相反，如果建模质量很好，则模型文件会变得很大，不仅占用更多的显存，也大大增加了载入模型和从系统内存向显存传输数据所需的时间，这样虽然效果变得更细致了，但过多的时间和空间的开销可能会影响程序的性能，比如在 3D 游戏中，如果场景和人物模型都很细致，那么性能一般的显卡可能不能进行实时渲染。有许多成熟的曲面技术[1–3]，如通用的细分算法[2]和曲面样条技术[3]，但它们需要存储临近三角面片的信息，计算和存储临近信息会产生较大的开销。

1 系统简介

在本文中，临近的三角形面片共用顶点位置和法线信息，渲染管道中每一个三角面片经过贝塞尔三角化在 GPU 上被细化成许多小的凸起的三角面片，从而产生了近似的曲面效果，使先前的三角形面片之间的过渡更加平滑。细分产生的三角形由 GPU 进行运算，过程类似于基于 GPU 的地形细分[4–5]，并且因为细分的结果在渲染管道内传输，甚至不需要额外的显存空间，且考虑到 GPU 运算的高并行性和快速性，PN 三角形曲面细分相对于精细建模更有空间和时间上的优势。以 DirectX 11 管线流程[6]为例(如图 1 所示)，细分的过程处在 Vertex 着色器和 Primitive 集成之间，包括 Hull 着色器、细分阶段和 Domain 着色器。Hull 着色器主要进行细分因子和控

收稿日期：2014-04-03；修回日期：2014-06-13

*通信作者：吴志红 email:wuzhihong@scu.edu.cn

制点的计算，细分阶段是固定管线阶段，由 GPU 自动完成，不支持编程，而最后的 Domain 着色器则计算细分后的顶点信息，交由下一阶段来集成。

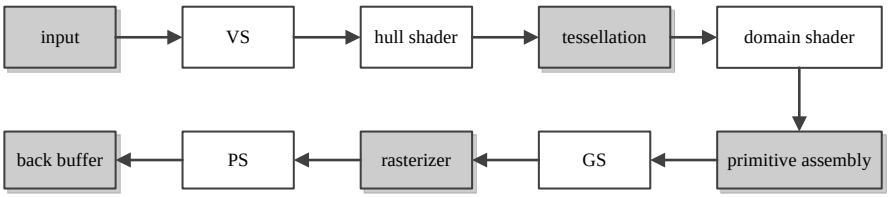

Fig.1 DirectX 11 pipeline flow (Those with gray shadows are fixed, others are programmable)
图 1 DirectX 11 管线流程(有灰色阴影的是固定管线，其他的是可编程管线)

2 曲面化的 PN 三角形

2.1 PN 三角形与贝塞尔三角形

贝塞尔三角形是一种特殊的贝塞尔曲面，它通过控制点和质心坐标信息来确定三次曲面上的点的位置[7]，而 PN 三角形又是贝塞尔三角形的一种特殊的实现，即 PN 三角形的控制点信息是依据输入三角形的顶点位置信息和法线信息计算求得，而它的质心坐标则是通过细分着色器来进行插值并输出。

2.2 几何控制点

输入渲染管线中的三角面片的信息以顶点为单位，如图 2 所示，$\boldsymbol{P}_1 \sim \boldsymbol{P}_3$ 是输入顶点的位置信息，$\boldsymbol{N}_1 \sim \boldsymbol{N}_3$ 是输入顶点的法线信息，基于这些信息可求得控制点信息。如图 3 所示，一个三角面片共有 10 个控制点，其中 $\boldsymbol{b}_{003}, \boldsymbol{b}_{300}, \boldsymbol{b}_{030}$ 是原三角形的 3 个顶点，而其余的 7 个控制点则是依据顶点和法线信息插入的，之后根据控制点信息和细分着色器输出的质心坐标信息共同计算出插入点的位置。

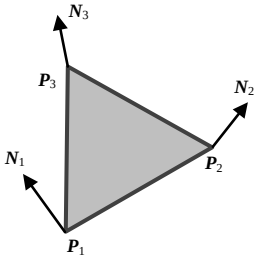

Fig.2 Input data, point and normal
图 2 输入数据、点和法线

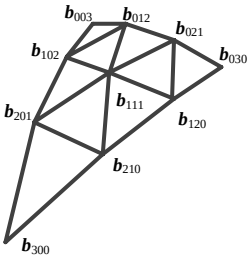

Fig.3 Control points of PN triangle
图 3 PN 三角形的控制点

2.3 控制点的计算

1) 如图 4 所示，$\boldsymbol{b}_{ijk}$ 所对应的三角面片上的插入点 $\boldsymbol{P}_0 = \dfrac{i\boldsymbol{P}_1 + j\boldsymbol{P}_2 + k\boldsymbol{P}_3}{3}$；

2) 距 $\boldsymbol{P}_0$ 点最近的顶点作为一个平面中的点，以该顶点的法线为该平面的法线；

3) $\boldsymbol{P}_0$ 点在上述平面上的投影即为所求的控制点 $\boldsymbol{b}_{ijk}$。

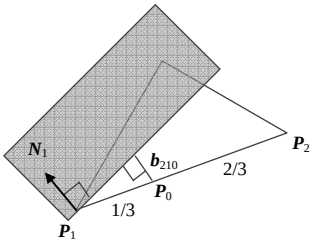

Fig.4 Sketch map of getting control points
图4 求控制点信息示意图

求 $\boldsymbol{b}_{210}$ 的过程为：首先计算插入点 $\boldsymbol{P}_0$，其中，i=2, j=1, k=0，之后找到点 $\boldsymbol{P}_1$ 和它的法线 $\boldsymbol{N}_1$ 所对应的平面，最后将点 $\boldsymbol{P}_0$ 投影到前述的平面上，即可得到控制点 $\boldsymbol{b}_{210}$。控制点的求法类似，而 $\boldsymbol{b}_{111}$ 则有些特别，求法为：$\boldsymbol{E} = \dfrac{\boldsymbol{b}_{210} + \boldsymbol{b}_{120} + \boldsymbol{b}_{021} + \boldsymbol{b}_{012} + \boldsymbol{b}_{102} + \boldsymbol{b}_{201}}{6}$，$\boldsymbol{V} = \dfrac{\boldsymbol{P}_1 + \boldsymbol{P}_2 + \boldsymbol{P}_3}{3}$，$\boldsymbol{b}_{111} = \boldsymbol{E} + \dfrac{\boldsymbol{E} - \boldsymbol{V}}{2}$。$\boldsymbol{b}_{111}$ 的值是 $\boldsymbol{E}$ 加上从 $\boldsymbol{V}$ 到 $\boldsymbol{E}$ 向量方向的偏移的结果。

2.4 法线控制点

PN 三角形共有 6 个法线控制点(如图 5 所示)，其中顶点处的 3 个为顶点的法线向量，而 $\boldsymbol{n}_{110}, \boldsymbol{n}_{011}, \boldsymbol{n}_{101}$ 则是插入的控制点信息，这些控制点信息和细分后的质心坐标共同影响着新顶点的法线向量的插入结果。法线向量的插入可以选择线性的，也可以选择二次的，鉴于 PN 三角形点的插入是三次的(即曲面的)，而法线向量又是对曲面方程求偏导数的结果，因而选法线的二次插入。

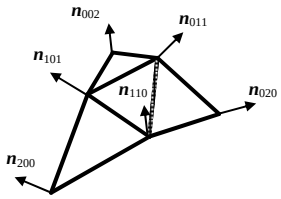

Fig.5 Sketch map of getting control points
图5 求控制点信息示意图

基于面法线的渲染中三角面片之间的法线过渡是离散的，所以需要插入法线，如果是简单的线性插入，所插入的法线信息可能不能反映法线反射的情况。比如，起始点和终止点的法线向量的方向相同，则在这之间插

入的所有法线向量的方向与起始点和终止点的方向相同，但实际情况可能是，起始点和终止点之间的曲线可能是类似于正弦曲线的形状，而在这种情况下，所插入的法线向量的方向显然不是全都相同的。

基于上述原因，在求法线控制点时考虑镜面映射，如图 6 所示，$\boldsymbol{n}_{110}$ 为向量 $\frac{\boldsymbol{N}_1+\boldsymbol{N}_2}{2}$ 相对于垂直于向量 $\boldsymbol{P}_1,\boldsymbol{P}_2$ 的平面的镜面映射，向量 $\boldsymbol{P}_1,\boldsymbol{P}_2$ 是上述平面的法线向量，但还没有规格化，有了平面上的点 $\frac{\boldsymbol{P}_1+\boldsymbol{P}_2}{2}$ 和平面的法线向量，$\boldsymbol{n}_{110}$ 很容易通过 $\frac{\boldsymbol{N}_1+\boldsymbol{N}_2}{2}$ 相对于平面法线的投影求出。设 $\frac{\boldsymbol{N}_1+\boldsymbol{N}_2}{2}$ 在 $\boldsymbol{P}_1,\boldsymbol{P}_2$ 上的投影为向量 $\boldsymbol{H}$，则 $\boldsymbol{n}_{110}=\frac{\boldsymbol{N}_1+\boldsymbol{N}_2}{2}-2\boldsymbol{H}$。

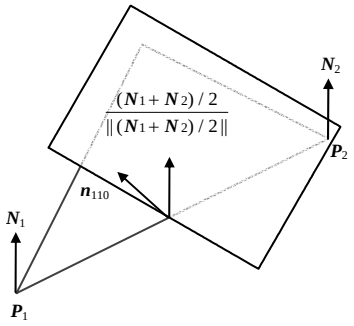

Fig.6 Reflection of normal control point
图6 法线控制点的镜面映射

3　曲面细分中的计算

3.1 顶点的插入公式

顶点的插入公式是三次贝塞尔三角形的计算公式：

$$\boldsymbol{P}(u,v,w)=\boldsymbol{b}_{300}w^3+\boldsymbol{b}_{030}u^3+\boldsymbol{b}_{003}v^3+3\boldsymbol{b}_{210}w^2u+3\boldsymbol{b}_{120}wu^2+3\boldsymbol{b}_{210}w^2v+3\boldsymbol{b}_{021}u^2v+3\boldsymbol{b}_{102}wv^2+3\boldsymbol{b}_{012}uv^2+6\boldsymbol{b}_{111}wuv$$

$\boldsymbol{P}(u,v,w)$ 是插入后的顶点位置，由 $(\alpha u+\beta v+\gamma w)^3$ 展开得来，多项式的各项系数用 $\boldsymbol{b}_{300}$ 到 $\boldsymbol{b}_{111}$ 代替，u 和 v 是细分所输出的坐标值，而 w=1–u–v，因为 u,v,w 是三角形的质心坐标，而 $\boldsymbol{b}_{300}$ 到 $\boldsymbol{b}_{111}$ 这 10 个控制点信息是沿基于顶点的法线方向“偏移”而得来的，因此 $\boldsymbol{P}(u,v,w)$ 落在该三角平面的外侧(即落在平面法线所指的半空间)，这样把求得的三角面片的插入点连成小的三角面片集合后的外观效果是凸起的(如图 3 所示)，如此便可在完成细分操作的同时，使一个三角面片更贴近于数学曲面。

考虑 2 个相邻的三角形，它们有 2 个公用的顶点和 1 条公用的边，因为法线信息是基于顶点的而非基于三角面片，因而这 2 个三角形也公用顶点法线信息。在使用置换贴图时，如果 2 个三角形在这条公用边上的细分因子不同，则可能产生裂缝的效果，比如一个三角形在这条边上的细分因子为 1(即在这条边上无细分)，而另一个为 3(即在这条边上再插入 2 个点)，则前者还在原来的平面，而后者因为上述公式的“矫正”而凸起，与前者错开，不能实现无缝拼接的效果。为了避免产生上述瑕疵，在计算细分因子时应当以边为基本单位，比如计算视点与边的中点之间的距离，这样，即便 2 个相邻三角形的质心到视点的距离不同，也不会产生裂缝效果[8]。

3.2 法线的插入公式

法线的插入公式是二次贝塞尔三角形的计算公式：$\boldsymbol{N}(u,v,w)=\boldsymbol{n}_{200}w^2+\boldsymbol{n}_{020}u^2+\boldsymbol{n}_{002}v^2+\boldsymbol{n}_{110}wu+\boldsymbol{n}_{011}uv+\boldsymbol{n}_{101}wv$。

$\boldsymbol{N}(u,v,w)$ 为插入后的法线向量，由 $(\alpha u+\beta v+\gamma w)^2$ 展开得来，多项式的各项系数由 $\boldsymbol{n}_{200}$ 到 $\boldsymbol{n}_{101}$ 代替，u,v,w 和顶点插入公式中的坐标信息一致，法线的控制点信息能够平滑地反映插入点的法线信息间的过渡，因而可以产生圆润的光照效果[9]。

4　LoD 计算

4.1 效率分析

因为效率问题，细分因子在整个模型空间不应是固定的，而应是自适应的，否则视点离物体较远时所渲染的三角面片数并未减少，同时观察者所看到的物体体积变小，细分的优势不仅体现不足，而且效率并不高。因而细分因子应该与视点和三角面片之间的距离成反比，这样当视点靠近物体时距离减小，LoD 增加。然而虽然观察者离物体越近时细分越多，但也应该设定一个最高限度，因为当细分达到一定程度时，继续细分下去对外观的影响微乎其微，但效率却大打折扣。

当视点快速接近物体时，由于 LoD 增加速率过快，物体的外形因为变化过程突出而有可能被观察者看出并被当作渲染的瑕疵，因此最好设置一个最低的细分限度，从而保证外形的变化之间有着平滑的过渡，而当观察者离物体距离较远并达到限定的值时，可以将细分因子设置成 1，即不进行细分。

4.2 LoD 公式

细分因子的计算公式为： $Factor = \mathrm{Min}(\mathrm{Max}(low,(length / distance) * ratio * controller * high))$。其中，Min 和 Max 分别是取最小和最大函数，*low* 和 *high* 分别是细分下限和上限，*length* 是三角面片的边长(在与视点距离相同的情况下，边越长则细分程度应越高)，*distance* 是视点与三角面片边的中点之间的距离，*ratio* 是现有缓冲区大小与初始缓冲区大小之比，它也影响着细分因子，比如窗口变大，*length* 和 *distance* 并没有变化，但每个三角面片所占屏幕上的像素点更多了，物体显得更大，因而需要增加细分程度。*controller* 是一个调控因子，可以设置成固定数值，也可以作为一个变量供用户调节。当 *length*/*distance* 小于某个特定的值，比如 0.015 时，*Factor* 设定为 1。自适应调节的结果如图 7 所示，当视点远离物体时，细分因子逐渐减小，效率增加，但外观上并无大的变化。

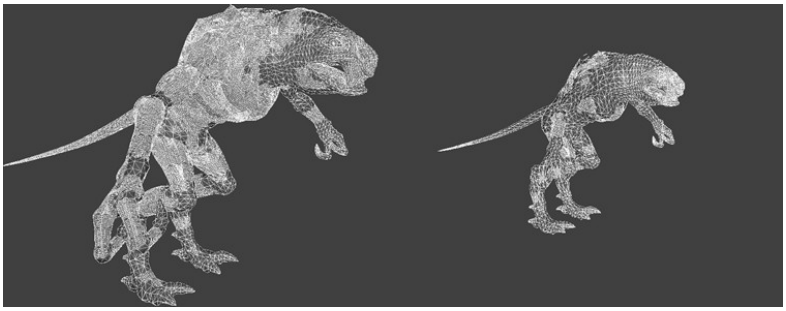

Fig.7 LoD based on distance between eye and object
图 7 LoD 随视点离物体之间距离的变化

4.3 效率对比和分析

测试环境为 Intel i3-2348M 处理器，NVIDIA GT635M 显卡和 DirectX 11。由图 8 的对比可以看出细分后的渲染结果更加柔和，外观上相比于细分前有很大改善。

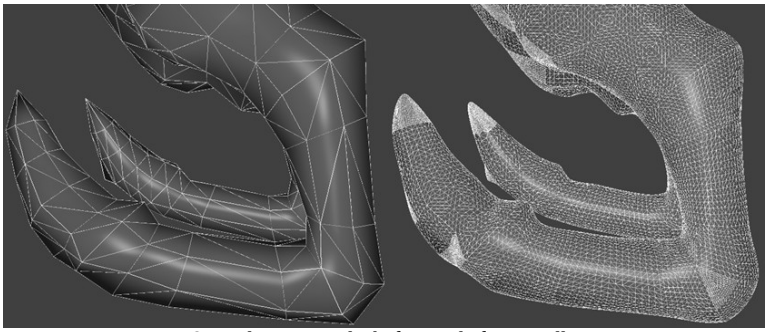

Fig.8 Renderging results before and after tessellation
图 8 细分前后渲染结果对比

设不使用 LoD 的情况为 A，而使用 LoD 的情况为 B，4.2 节中 *low*=2, *high*=15, *ratio*=1, *controller* 为表 1 的第 1 列，视点坐标适中(全部模型都在视点范围内)，则 2 种情况的采样结果如表 1 所示。

表 1 统计信息采样表
Table1 Statistics

controller factor	A triangles	B triangles	A frames/s	B frames/s	triangles A/B	frames A/B
1	21 044	21 044	237.25	236.77	1.00	1.00
52	1 079 828	36 478	132.34	232.85	29.60	0.57
100	3 726 788	62 802	55.35	231.79	59.34	0.24
150	7 961 924	103 654	27.50	224.66	76.81	0.12
200	13 785 236	155 464	15.98	205.90	88.67	0.08
253	22 587 392	220 566	9.95	192.95	102.41	0.05
300	30 196 388	294 856	7.43	185.32	102.41	0.04
348	40 784 228	383 514	5.48	177.34	106.34	0.03
403	55 145 000	505 594	4.09	168.93	109.07	0.02
454	69 173 888	618 732	3.28	163.22	111.80	0.02
512	87 547 188	769 224	2.58	158.15	113.81	0.02

随着细分因子(公式中的 *controller*)逐渐增大，A 对 B 的面片数比达到了 100 以上，相反，帧比率却一直减小，最低仅为前者的 1%左右(表中的 0.02 是四舍五入的结果)，以上数据说明，如果不使用优化的 LoD，不仅处理了大量的冗余数据，而且效率很低，难以在实时渲染中应用。如果使用优化的 LoD，则可以极大地减少数据的处理量，并且提高程序的计算效率，同时也能达到所需的渲染目的。

上述数据对细分因子设定了上限，当取消上限设定，并且 *controller* 不断增大时，三角面片数增加的速率要比 FPS(Frames Per Second)减小的速率大，如表 2 所示，而且即便最后细分级别达到 500 左右，每帧处理的三角面片数高达 2 367 235 个，但依然能够达到 43 FPS，这对于实时程序来说已经足够。

表 2 效率分析表

Table2 Analysis of performance

LoD	1	50	100	150	200	250	300	350	400	450	500
triangles	7 239	41 254	117 150	243 794	418 933	619 721	912 354	1 194 147	1 527 369	1 923 607	2 367 235
frames	315	306	268	210	150	122	90	76	62	55	43

5　结论

PN 三角形不仅对现有的模型没有更多的要求(仅能由三角形或四边形表示，相比于实时光线跟踪技术[10]，后者则能处理更多的几何表示)，而且效率很高，可以应用于实时的程序中。当细分因子大于某个点之后，模型的外观产生的变化很小，因而在实际应用中不需要对模型进行较大的划分便可以产生足够饱满的渲染效果，因而可以用于大场景的渲染中，提高整体场景的感官效果。

参考文献：

[1] Vlachos A,Peters J,Boyd C,et al. Curved PN triangles[C]// Proceedings of the symposium on interactive 3D graphics. North Carolina,USA:ACM Special Interest Group on Data Communication, 2001:159–166.

[2] Charles T Loop. Smooth subdivision surfaces based on triangle[D]. Salt Lake City,Utah,USA:University of Utah, 1987.

[3] Peters J. Smoothing polyhedra made easy[J]. ACM Transactions on Graphics, 1995,14(2):161–169.

[4] Hyeongyeop K,Junghyun H. Multi-resolution terrain rendering with GPU tessellation[J]. The Visual Computer, 2014, 31(4):455–469.

[5] MU Xiaodong,NIU Xiaolin,ZHANG Tong,et al. Terrain rendering using GPU tessellation[C]// Advanced in image and graphics technologies communications in computer and information science. Beijing,China:Springer, 2013:269–276.

[6] Frank D Luna. Introduction to 3D Game Programming with DirectX 11[M]. [S.l]:Mercury Learning and Information, 2012:473–480.

[7] Bezier triangle[EB/OL]. [2014-04-03]. http://en.wikipedia.org/wiki/B%C3%A9zier_triangle.

[8] Tamy Boubekeur,Alexa M. Phong Tessellation[C]// Proceedings of SIGGRAPH Asia. Suntec City,Singaporea:ACM SIGGRAPH, 2008:1–4.

[9] Hanyong J,Junghyun H. Feature-preserving displacement mapping with Graphics Processing Unit(GPU) tessellation[J]. Computer Graphics Forum, 2012,31(6):1880–1894.

[10] Parker S,Bigler J,Dietrich A,et al. Optix: a general purpose ray tracing engine[J]. ACM Transactions on Graphics 2010, 29(4):1–13.

作者简介：

陈　驰(1988–)，男，河南省商丘市人，在读硕士研究生，主要研究方向为计算机图形学.email:gamesleonchen@gmail.com.

吴志红(1964–)，女，北京市人，博士，副教授，主要研究方向为智能系统与信息处理.