樹木のアニメーション及びレンダリングにおける LOD 手法の適用

近藤　亮 †　宮島　信吾 †　大出　剛史 †　金井　崇 †
† 慶應義塾大学 環境情報学部
E-mail: {t00379rk/t01924sm/t00144to/kanai}@sfc.keio.ac.jp

樹木のモデリングに関する研究は数多く行われており、リアリティの高い樹木形状が半自動的に得られる。一方、樹木を含む大規模なシーンのインタラクティブなレンダリングにおいて臨場感を得るには、風による樹木の動きの計算を考慮する必要があるが、樹木を構成する形状プリミティブの数が多い場合には、フレーム毎の計算コストが非常に高い。このことがインタラクティブなレンダリングの障害となっている。そこで本研究では、 LOD の考え方を適用し、アニメーションやレンダリングの制御を動的に行う方法を提案する。

Applying LOD Techniques for Animation and Rendering of Trees

Ryo Kondo †　Shingo Miyajima †　Takefumi Ohide†　Takashi Kanai †
Keio University, Faculty of Environmental Information
E-mail: {t00379rk/t01924sm/t00144to/kanai}@sfc.keio.ac.jp

Thanks to the researches about modeling of trees, realistic tree models with textures can be semi-automatically created. On the other hand, the motion of trees according to the wind should be calculated to get more realistic presence in large animated forest scenes. If the number of primitives in a tree is large, per-frame dynamics calculation causes high computational costs, which prevents interactive rendering. In this paper, we propose two methods for the speedup of tree animation and rendering. One is based on mainly using GPU, the other uses CPU only with LOD techniques. We also compare such two methods in several large forest scenes.

1　はじめに

樹木や植物を含んだ自然のシーンを生成しレンダリングする試みは、これまでも続けられてきた。その際に問題となるのが、樹木など自然物のオブジェクトの形状の複雑さとその計算コストである。一般的に、一本の樹木を構成するには数万から数十万のポリゴンが必要となる。樹木が生い茂るシーンなどでは、合計数千万ポリゴンに及ぶことも少なくないため、レンダリングのプリミティブを線や点に置き換えたり、一本の樹木のデータをレンダリング時に複数の木で再利用してデータ量を抑えるなどの手法が提案されてきた。

一方、リアルで動的な樹木のシーンを考慮した場合、風による枝のアニメーションを考慮する必要があるが、先述したとおり全てのポリゴンに対するアニメーションによる座標計算を CPU で計算するには計算コストが高すぎる。そこで、本論文では LOD の考え方を元にして必要なポリゴンに対してのみ座標計算を CPU で行う手法と、全て GPU を用いて計算させる手法の二つの長所・短所についてそれぞれ検証を行う。

2　関連研究

2.1　樹木の形状生成

樹木の形状を生成するためには、最初に軸となる幹や葉の生える位置・方向を決定し、次に幹としての太さを持たせる方法が一般的である。これらの位置・方向は植物種や環境から受ける影響を元にパラメータ化される。それらに関する研究はこれまで数多くなされてきた。

例えば Weber と Penn による研究では、枝を再帰的に取り扱い、その再帰の深さ毎に分枝の角度などのパラメータを設定することで非常に写実的な形状を再現している [5]。また、 Soler らの研究では、光の放射エネルギーを元に植物の生長をシミュレートし、日光に向かって成長する植物の様子を再現している [8]。 Prusinkiewicz らの最新の研究では、実際の植物を位置情報に基づいてパラメータ化するための手法を提案しており、バリエーションに富んだ植物の再現に成功している [7]。

2.2 樹木のレンダリング

Weber と Penn による研究では、距離に応じてポリゴン・ライン・ポイントとレンダリングのプリミティブを変化させてレンダリングコストの削減を図る手法が提案されている [5]。一例では、最も近い距離で 7 万ポリゴンの樹木が、最も遠い場合では 2 千ポイントにまで削減されている。また、 Deussen らによる研究ではその手法を拡張し、シーン全体の植物の配置を考慮し、似た植物をグルーピングすることでデータを再利用し、大幅な削減を行う手法を提案している [6]。

以上はオフラインレンダリングによる研究であるが、その後の同グループによる研究では、グルーピングされた植物の重要度を考慮してレンダリングする際のプリミティブを変化させ、実際に OpenGL による実装のための最適化の手法を提案している [3]。その結果、今日の一般的なパーソナルコンピュータとビデオカードを用いて、インタラクティブなフレームレートでエコシステムのレンダリングを行う可能性が示された。

2.3 ビルボーディング・イメージベースドレンダリング

従来のビデオゲームなど高速な処理が必要とされる場面では、樹木はビルボードとして取り扱われてきた [2]。最も単純な方法では二枚のポリゴンを直交して配置することで表現する。他にも IDV 社の開発した SpeedTreeRT[1] では、一枚または複数枚の葉を一つのビルボードに見立て、より現実感のある樹木が表現されている。ただしビルボードの性質上、樹木を頂上付近から見下ろした際に起こる葉の不自然な回転は未だ解決されていない。

イメージベースドレンダリングのアプローチとしては、樹木を複数の断面に予めレンダリングしておき、それらをテクスチャとして扱う手法が Jakulin らにより提案されている [4]。リアルタイム処理の際には、それらのテクスチャを用いて α ブレンディングで補完しながらレンダリングすることで全方向からのイメージの生成を実現している。

2.4 GPU

ここ数年間で最も目覚しいハードウェアの進化による技術として、 GPU を用いたプログラマブルシェーダが挙げられる。従来では頂点の座標計算は全て CPU が行っていたが、最近のビデオカードに搭載された GPU の頂点プログラムの機能を用いることで、毎頂点の座標計算をハードウェアで高速に処理することができるようになった。またプログラマブルという名前の示す通り、頂点毎に行われる計算手順をユーザが独自に書き変えることも可能となった。これに伴い、骨格を持ったキャラクタに対する頂点のスキニング処理など、様々なアニメーションに対応する頂点計算を高速に行うことができる。

Shape	樹木のシルエット (3.1節参照)
Basesize	枝の生えない部分の割合
0CurveResolution	主幹 (Level0) のノードの数
0Curve	主幹の曲がる度数
1Branches	主幹から発生する枝の数
1BranchLength	枝の主幹に対する長さ
1AngleDown	枝の主幹に対する分枝角度 (X)
1AngleRotate	枝の主幹に対する分枝角度 (Y)
1CurveResolution	枝のノードの数

表 1: パラメータとその内容

3 樹木の形状生成

ここでは、本研究に用いられた Weber らによる樹木生成の実装について説明する。

一本の樹木は複数の枝と葉から成り、さらに一本の枝は複数のノード (節) を持つ。枝はノード単位で少しずつ方向を変化させることで曲がり・しなりを表現している (図 1)。

枝は分枝の深さによりレベルが決定される。主幹は通常 Level0 から始まり、主幹から発生する枝が Level1 さらにその枝から発生する枝は Level2 となる。レベルごとの分枝角度などのパラメータは、植物の種類により変化する。サンプルとして幾つかの植物のパラメータのデータファイルが公開されている[1]。

また、同種の植物でも個体ごとにばらつきがあり、全く同一のものは存在しない。そこで、個体間に生じるばらつきの度合いを表すパラメータが用意されている。

代表的なパラメータの一部とその内容を表 1に示す。

3.1 枝の発生・樹木全体の形状

枝の生成時に用いられるパラメータは、主に分枝角度・枝の長さ・半径である。分枝角度には AngleDown と AngleRotate があり、親ノードでの座標系での X 軸方向、 Y 軸方向への回転角度を表する。親ノードの座標系とは、枝が発生する位置のノードが持つ座標系であり、そのノードの伸びている方向を Y 軸、それに対する直交座標を X 軸とする (図 1)。また、 AngleRotate は一つ前の分枝に対する相対角度である。仮に Level1 の AngleRotate が 120 度であったとすると、 Level0 から発生する 4 本目の枝は主幹の軸方向に対して 1 周することになる。通常は 140 度前後の値をとり、一見してまばらに分散するようになる。

枝の長さは、親枝に対する枝の発生する位置 (オフセット値) に因る。オフセット値は親枝の全体の長さ

[1] URL: http://www.imonk.com/jason/viewtree/

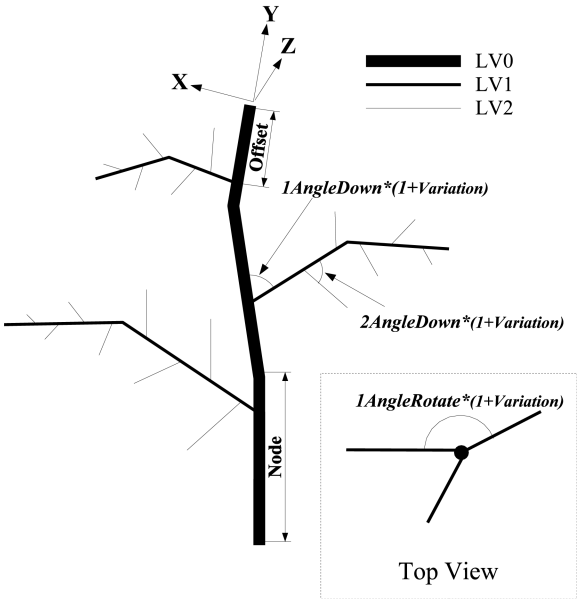

図 1: 枝の発生と角度

Shape	返り値
0	0.2 + 0.8 * Offset
1	0.2 + 0.8 * sin(π * Offset)
2	0.2 + 0.8 * sin(0.5 * π * Offset)
3	1.0
4	0.5 + 0.5 * Offset
...	...

表 2: shapeRatio 関数

を 1.0 としたときの、親枝の先端から枝の発生する位置までの長さをとる。特に親幹から発生する Level1 の枝の長さはその樹木の大体のシルエットを決定するため、Shape というパラメータを用いてオフセット値と枝の長さの関係を定義する (式 1)。Weber らの手法では Shape は 0 から 7 の整数値を取り、それぞれに対して関数が用意されている (表 2・図 2)。

$$Length_{\mathrm{Lv1}} = 1\mathrm{BranchLength} * Length_{\mathrm{Lv0}} * \mathrm{shapeRatio}(\mathrm{Shape}, \mathrm{Offset}) \quad (1)$$

3.2 葉の生成

葉も幹同様のパラメータを用いて生成する。例外として、AngleRotate が負の値を取った場合、その枝の葉は同じ位置 (オフセット) から常に複数枚単位で出現する。そうすることで互生・対生・輪生の葉を表現する (図 3)。このとき、同時に出現する枚数は

$$360/(-\mathrm{AngleRotate}) \quad (2)$$

で表される。

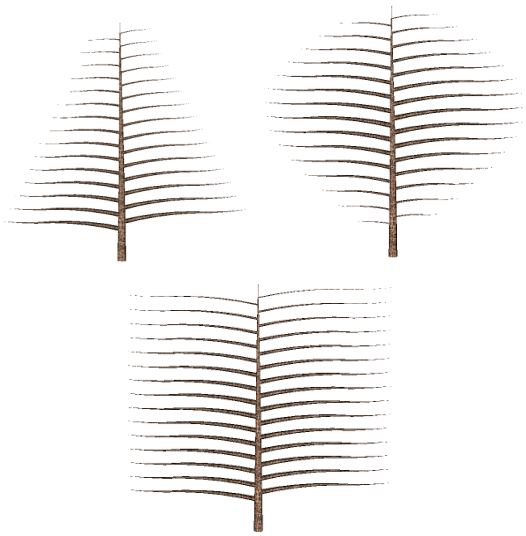

図 2: Shape=0,1,3 の例

4 アニメーション・レンダリング

ここでは上述の手法によって生成された樹木に対して風による枝の運動計算によるアニメーション及びレンダリングを行う手法を説明する。本手法では、枝を構成するノード間の角度を変化させることで風による揺れ・しなりを表現する。葉は枝のノードに対して固定であるとし、ノードから独立して振動はしない。ただし、葉柄をごく短い分枝とみなし、その枝から発生する葉の数を 1 枚にすることで同様の条件を満たす植物種を設定することは可能である。

4.1 運動計算

今回の実装では簡略化したばねモデルを用いた。一本の枝に対してかかる風による力 Fw は全て葉が風から受ける力であると見なすものとする。風力ベクトルを W、葉の総面積を A とすると

$$Fw = W \cdot A \quad (3)$$

と近似することができる。また、自重による力ベクトル Fg を求めるには、枝の長さ、太さ、密度をそれぞれ l, r, d と置き、葉の厚みを均一と考えたときの面積辺りの密度を Ag とすると、重力ベクトルを G としたとき

$$Fg = G \cdot (l * r * d + A * Ag) \quad (4)$$

と表す。l, r, d, A, Ag, g は固定なので、定数 Fg を用いることができる。以上より、一本の枝に対して外部からかかる力ベクトルは

$$F = Fw + Fg = W \cdot A + Fg \quad (5)$$

図 3: 4 枚の輪生の葉

で求められる。各枝に対し、その枝とそれ以降の子枝の F を合計したものを用いて、ノード毎に半径から回転角を求める。

4.2 枝のスキニング

4.1節によって求められたノード情報を元に、 GPU の頂点プログラムを用いて枝のスキニング及びレンダリングを行う。 OpenGL 1.x で頂点プログラムを利用する場合、 96 個の定数レジスタ (レジスタ 1 つにつき 4 つの 32 ビット浮動小数点値を持つ) が利用可能である。本手法では 1 つのノードを格納するのに 4 レジスタを用い、始点座標 (x, y, z)、半径、回転行列 (3x3) を格納する。また、レベル毎の頂点数はノードの数 (表 1の CurveResolution) に比例するので、同じレベルの枝の頂点数 (ポリゴン数) は等しい。そこで、ディスプレイ・リストを用いて頂点座標を与えることができる。

4.3 葉の頂点計算

葉の頂点計算は、枝と同様大量の計算が必要となる。試行錯誤の結果、 CPU で処理する場合と GPU で処理する場合の、二通りの手法が認められた。それぞれの特徴を分けて説明する。

4.3.1 GPU を用いた高速化

頂点プログラムが利用できる環境では、葉の座標計算を GPU で行うことが可能である。枝をスキニングした時点で定数レジスタにはノード情報が入っているため、その直後に葉のレンダリング処理を行う。

1 枚の葉がその枝に対して持つ定数値は、枝の根元からの位置 (オフセット値)、 RotateAngle、 DownAngle の 3 つの浮動小数点値である。葉を長方形で表した場合頂点は 4 つ必要であり、それぞれに対し上の定数値は共通であるので、定数値は法線として 1 度与え、頂点座標を 4 つ別に与える。頂点には長方形を Z=0 の 2 次元の平面に置いたときの X,Y の座標を与える。葉の長方形の横の長さを lx, 縦の長さを ly と置くと、 4 点は $(-\frac{lx}{2}, 0), (\frac{lx}{2}, 0), (-\frac{lx}{2}, ly), (\frac{lx}{2}, ly)$ となる。なお、 4 点が共通の法線を使用するため、シェーディングモデルには GL_FLAT を指定する。以下に擬似コードを示す。

```
for all Branch
 glNewList(Branch->ListID, GL_COMPILE);
 for Leaves in Branch
   glNormal3d(Leaf->Offset,
   Leaf->downangle, Leaf->rotateangle);
   glVertex3d(-Leaf->lx / 2, 0,0);
   glVertex3d(-Leaf->lx / 2, leaf->ly, 0);
   glVertex3d( Leaf->lx / 2, leaf->ly, 0);
   glVertex3d( Leaf->lx / 2, 0, 0);
 end for
 glEndList();
end for
```

頂点プログラムでは、法線情報に格納された定数値と定数レジスタに格納されたノード情報から葉の中心・姿勢を表す座標系を求め、与えられた頂点座標 $(x, y, 0)$ を葉を構成する長方形の 4 点の座標に変換する。しかしながら、これらの手法には冗長な点も幾つかある。葉を構成する 4 点が用いる座標系は共通であるのに、今回の実装では 4 つの頂点が与えられるたびにその座標系を計算している。これを解決する方法としては、定数レジスタに座標系を格納しておく方法がある。

ただし、現在のハードウェアの仕様では定数レジスタが 96 個しか存在しないことが障害となっている。サンプルデータでは枝に最大 20 ノードを持つ植物種が定義されているが、 4.2節で述べたように 1 つのノードにつき 4 つのレジスタを必要とする。つまりノード情報に 80 個の定数レジスタが占有され、葉には僅か 16 個しか割り当てられない。さらに葉の座標系は 4x4 行列なので、ノード同様 1 枚につき 4 レジスタ必要となる。つまり、 20 ノードの枝では最高 4 枚の葉しか処理できないことになる。

これを解決する糸口として、 nVIDIA 社から発売される NV30 を利用する必要があるだろう。定数レジスタは 256 個まで拡張されており、最大 20 のノードと 40 枚の葉を持つ枝を表現できることになる。 NV30 に限らず、各社ハードウェアベンダーは定数レジスタの拡張を行う方向にあるため、近いうちに普及するであろう。

同時に、葉の座標系を定数レジスタとして与えることで頂点プログラムの命令数を大幅に下げることができる。現在の実装では 90 前後の命令が必要であり、

リアルタイム処理には冗長と言わざるを得ない。実際、幹のスキニングと比較すると、1 ポリゴンに要するレンダリングコストが非常に高い。

4.3.2 LOD による CPU を用いた高速化

CPU を用いて葉の頂点計算を行う場合、頂点配列を用いる。Level1 の全ての枝は、自身とそれ以降の子枝が持つ全ての葉を格納するための頂点配列を確保する。各枝がスキニングされた際、そのノード情報を元に葉の座標を計算する。葉は枝のノードに対して固定であるので、枝が変化したときのみ座標計算を行えばよい。変化しない枝の葉は以前求めた座標を用いてレンダリングを行う。全ての葉の頂点配列を確保しておくことで、以前求めた座標がそのまま残るため再利用に都合が良い。

一方、枝は完全に静止することはほとんどないため、葉の座標計算の頻度を抑えるために枝が一定距離以上動いたかどうかで判定する必要がある。そこで LOD の手法を用いて、カメラから見たときの枝の移動量に従い葉の更新頻度を決定する。各枝ごとに移動量を求め蓄積し、それらがカメラからの距離に対し閾値を越えたとき、葉の座標更新を行い蓄積を 0 に戻す。これにより、遠方にある樹木など動きによる画像の誤差が生じにくい部分の計算コストを大幅に削減することができる。

閾値は蓄積された移動量、その枝と視点の距離、スクリーンのピクセル数、視野角、誤差係数により決定する。それぞれが $P, d, width, fovx, i$ のとき、葉の座標更新の条件式は

$$\frac{P}{d} > i * \frac{\tan(fovx)}{width} \tag{6}$$

と表せる。

5 結果・考察

4.3節で示した実装により 4 種類の樹木を用いてパフォーマンスの検証を行った。その結果を表 3に示す。プログラムの実行には、Pentium4 2GHz(物理メモリ 1024MB), GeForce4 Ti4600(ビデオメモリ 128MB) を用い、それぞれ 30m と 130m の距離からおよそ 20m 四方の樹木のシーンを見たときを例とし、GPU による高速化、CPU による高速化、および高速化なしの場合について、フレームレートを計測した。

5.1 CPU と GPU の違い

GPU による高速化と CPU による高速化では、樹木の数と距離に対するパフォーマンスの変化に違いが見られる。GPU による手法では、計算コストは枝と葉の数に比例し、距離に対して一定である。一方、CPU による高速化の手法では、計算コストは枝と葉の数に比例するが距離に対し反比例する。また風向きが大きく変化したとき、枝の移動量が急激に増加するため一時的にパフォーマンスが低下する。一般的には、樹木がカメラの近くに集中しているような状況では GPU による処理が適しており、樹木がカメラから遠い位置に集中するような状況では CPU による高速化が適している。

	樹木 A	樹木 B	樹木 C	樹木 D
葉 (枚)	57726	76441	37113	17150
枝 (本)	15614	1412	2153	503

距離	CPU	GPU	高速化なし
樹木 A 30m	7	5	4
130m	7-8	5	4
樹木 B 30m	11-14	13	11
130m	13-20	13	11
樹木 C 30m	14-15	15	13
130m	17	15	13
樹木 D 30m	36-37	38	35
130m	37-41	38	35

表 3: 樹木と距離によるパフォーマンスの比較 (fps)

5.2 問題点

現在の実装における問題点として、まず 4.3.1節で述べたような GPU の命令数の肥大化と無駄な処理の繰り返しが挙げられる。これは間もなく登場するハードウェアを利用することで解決される。次に、現在の実装ではレンダリングされるプリミティブは全て四角形 (GL_QUAD) であるので、ポイントなど低コストのプリミティブを用いることでパフォーマンスを向上させる必要がある。これらについては 2.2節で紹介した過去の研究が非常に参考になるだろう。

6 今後の課題

本文では樹木のアニメーションをインタラクティブに行う手法を紹介し、実際に PC 上で動かせることを提示した。これらに過去の研究を組み合わせることで、さらに現実的なパフォーマンスを得ることができるだろう。また、GPU による頂点プログラムの命令数を削減することができれば、葉に対する振動など新たな表現を GPU で表現することができる。

また、現在ピクセルシェーダを用いたリアルタイム処理での複雑な光の表現に関する研究が盛んである。それらを適応することで、よりリアルな葉の表現も可能となる。

図 4: 実行例

参考文献

[1] *SpeedTree RT*. IDV Inc. http://www.idvinc.com/speedtree/.

[2] Akenine-Möller, T., and Haines, E. *Real-time Rendering*. A K Peters, Ltd, 63 South Avenue Natick, MA 01760, 2002, ch. 8.3.3 Axial Billboard, pp. 324–325.

[3] Deussen, O., Colditz, C., Stamminger, M., and Drettakis, G. Interactive visualization of complex plant ecosystems. In *Proceedings of the IEEE Visualization Conference* (2002), IEEE.

[4] Jakulin, A. Interactive vegetation rendering with slicing and blending. In *EUROGRAPHICS 2000 Short Presentations* (2000).

[5] J.Weber, and J.Penn. Creation and rendering of realistic trees. In *Computer Graphics (Proc. SIGGRAPH 95)* (1995), ACM Press, New York, pp. 119–128.

[6] O.Deussen, P.Hanrahan, M.Pharr, B.Lintermann, R.Mech, and P.Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *Computer Graphics (Proc. SIGGRAPH 98)* (1998), ACM Press, New York, pp. 275–286.

[7] Prusinkiewicz, P., Muendermann, L., Karwowski, R., and Lane, B. The use of positional information in the modeling of plants. In *Computer Graphics (Proc. SIGGRAPH 2001)* (2001), ACM Press, New York, pp. 289–300.

[8] Soler, C., Sillion, F., Blaise, F., and Dereffye, P. A physiological plant growth simulation engine based on accurate radiant energy transfer. Tech. Rep. 4116, INRIA, IMAGIS / LIAMA, February 2001. 31 pages.