2018 年度　卒　業　論　文

ジオメトリシェーダ上での制限を考慮した樹木形状の自動生成に関する研究

指導教員：渡辺 大地 准教授

メディア学部 ゲームサイエンス プロジェクト
学籍番号　M0115350
山本　馨加

2019 年 2 月

2018 年度　　卒　業　論　文　概　要

論文題目

ジオメトリシェーダ上での制限を考慮した
樹木形状の自動生成に関する研究

メディア学部 学籍番号：**M0115350**	氏名	山本　馨加	指導教員	渡辺 大地 准教授

キーワード	樹木、枝生成、ジオメトリシェーダ、自動生成、GPU、処理負荷

近年、オープンワールドのゲームでは広大なステージを制作するため、植物のモデルを１つ１つ手作業で配置するとステージの制作に膨大作業時間がかかってしまう。そのため、自動生成を用いることが多い。植物や木の自動生成の分野では様々な研究が行われているが、ゲームなどのリアルタイムに処理を行う場合、快適なプレイを必要とする。そのため、クオリティの高い表現をするだけではなく、CPU やメモリの使用量を調整して制作をしなければならない。

本研究では樹木形状の生成を CPU ではなく GPU で行い、ジオメトリシェーダを使うことによる処理負荷の軽減に着目した。ジオメトリシェーダは新たに頂点を 128 頂点まで生成することができるシェーダである。本研究では GPU 内でジオメトリシェーダを使い樹木形状を生成することを想定し、CPU 内で制限や特徴を考慮したアルゴリズムの提案を行う。樹木形状を 128 頂点のみを使い生成することは難しい。そのため、パーツごとに分割をして生成を行うことで、128 頂点の制限内で自動生成を可能とした。また、GPU は計算処理を非同期で並列的に行っているため、逐次処理を前提とした従来の樹木形状の生成を GPU に適用することはできない。本手法では枝を生成するときに軸となる枝を決めその軸を基準に枝を生成していくことで、非同期で並列的な処理を可能とした。木の幹は事前に用意したモデルを使い、樹木の枝を自動生成し、枝の形状を変えることで樹木形状を表現する。そのため、本研究では枝の生成に焦点をあてた手法を提案する。CPU 上でジオメトリシェーダの制限内で樹木形状の自動生成を行うことができ、任意の枝の形状を生成することができた。与える数値によって自己交差が起こってしまう場合がある。

目 次

図 目 次

第 1 章

はじめに

1.1　研究背景と目的

近年 PC やゲーム用ハードウェアなどの性能が向上し、クオリティの高い CG を使った映像作品やゲームなどが多く存在している。町や森林などの風景が綺麗に表現されており、リアリティの高い作品やアニメや漫画のようなイラスト風の作品など様々なコンテンツでグラフィックの表現の幅が広がっている。近年のゲームでは、自動生成により綺麗な森林を作成したものが多く存在している。特にオープンワールドのゲームでは広大なステージを制作するため、植物のモデルを 1 つ 1 つ手作業で配置するとステージの制作に膨大作業時間がかかってしまう。そのため、Horizon[1] や Witcher3[2] など、広大なステージを必要とするゲームでは地形生成に自動生成を用いることが多い。Horizon のステージは Jaap[3] らが地形データを元に植物のモデルの自動配置をしてステージの制作を行った。

植物や木の自動生成の分野では、竹内ら [4] の都市特有の条件を考慮し樹木の生長シミュレーションを行った研究や溝口ら [5] の表面の成長による樹木の形状生成を行った研究や大志ら [6] の樹木の CG の肥大生長シミュレーションを行った研究、溝口 [7] の階層構造を考慮し木の生成分布に関しての研究などがある。しかし、ゲームなどのリアルタイムに処理を行う分野ではクオ

リティの高い映像を表現するだけではなく、処理速度なども考慮に入れて制作をしなければならない。

オープンワールドのゲームでは、森林など自然の表現のために植物のモデルを多く配置し、クオリティの高い森林が表現されている。FinalFantasy15[8][9] では木や雑草などの植物のモデルを作り、それらを配置することで森林などのステージを作っている。そのようなステージなどは沢山のモデルを配置するため、キャラクタの攻撃などによる動的なモデルの編集などには CPU やメモリの容量を多く使う。しかし、リアルタイムレンダリングを行うコンテンツでは、グラフィックの他にサウンドやシステムなどにも CPU やメモリを使うため、他の要素と調整を行い実装する必要がある。また、ゲームでは描画の遅延が起こらないようにし、快適にプレイできる環境が必要である。快適なプレイを実現するには、サウンドとシステムとグラフィックで CPU やメモリの使用率を調整し、グラフィックに使用できる CPU やメモリ内でクオリティの高いグラフィックの表現をしなければならない。そのため、ゲームなどのリアルタイムに処理を行う分野で処理負荷を考慮したモデルの自動生成が必要となる。本研究では CPU で行っていた樹木形状の生成を GPU で行うことで樹木形状の自動生成に関して処理負荷を軽減することに着目した。

GPU[10][11] は非同期で並列的に多くの計算処理を同時に行うことで、膨大な計算処理を素早く行うことができるプロセッサである。そのため、多くの計算処理を行う場合や 3D モデルなどを使って画面に描画を行う計算処理に GPU を用いる。3D モデルを使ったゲームなどでは画面に表示する際、描画に必要な多くのモデル情報などを CPU から GPU へ送り、GPU 内で計算を行い画面に表示を行う。GPU 内でモデルを生成することで CPU から GPU へ情報を減らすことが可能である。GPU で樹木形状の生成処理の負荷を軽減する方法としてジオメトリシェーダ [12][13][14] というものに着目した。

ジオメトリシェーダとはプログラマブルシェーダの 1 つであり、GPU で新たに頂点を 128 頂点まで生成可能なシェーダである。このシェーダを使い、赤木ら [15] は形状生成に基づく樹木アニ

メーションの高速化を行い、柴原ら [16] は物理シミュレーションの研究を行い、奥屋ら [17] は投影面上での曲がり具合いを考慮した輪郭線描画の研究を行った。いくつかの頂点を新たに生成することで点や線、三角形などのモデルを構成するプリミティブな形状を編集でき、新たなモデルの生成やモデルのポリゴン形状を変更することができる。本研究は、GPU 内でジオメトリシェーダを使い樹木形状の生成を想定とした、CPU 内での樹木形状の生成アルゴリズムの提案を行う。

樹木形状を 128 頂点のみを使い生成することは難しく、樹木の形状生成する時に工夫する必要がある。本手法では 128 頂点内で樹木形状を生成するために、樹木形状を 128 頂点内で生成できるパーツごとに分割をして生成を行う。生成した各パーツを全て合わせて 1 本の樹木形状を生成することで、128 頂点の制限内で自動生成を可能とした。

また、GPU の特徴も考慮する必要がある。木の自動生成で有名なアルゴリズムとして、L-system というものがある。Aristid Lindenmayer 氏が考案したアルゴリズムで、自然界にある植物の成長や複雑な形状を表現するときによく用いる。L-system を使うことで、自分が設定した規則に沿った植物の形状や幾何学的なものを作ることができる。以下の図 1.1 は L-system を使って樹木を自動生成した例である。

図 1.1　L-system を使って樹木を自動生成した例

このアルゴリズムを使い、武田ら [18] はズーム動作と階層構造による新しい読書体験の研究を行い、大西ら [19] は生長シミュレーションによる 3 次元樹木モデルの生成の研究を行い、進藤ら [20] は生長シミュレーションによるツタの CG モデル生成の研究を行った。

L-system の様な従来の樹木形状の自動生成手法は、逐次処理を前提としたアルゴリズムである。しかし、上記で述べた通り GPU は非同期で並列的に計算処理を行っている。そのため、逐次処理を前提とした従来の樹木形状の生成を GPU に適用することはできない。本手法では枝を生成するときに軸となる枝を決め、その枝を基準に枝を生成していくことで非同期で並列的な処理を可能とした。

ゲームなどの樹木形状の生成では、葉はテクスチャ、木の幹は事前に用意したモデルを使い表現を行う。また、樹木形状を表現する上で樹木の形を一番表すのは枝の生成である。そのため、本研究では枝の生成に焦点をあてた手法を提案する。

本手法を用いたプログラムに任意の数値を与え、枝の生成を行った。3 つのプログラムを実行

し、与えた数値を元に枝となる各パーツの生成を実行した。その結果として、3 つのプログラムが別々に生成した枝を合わせた時、想定していた枝の形状になった。枝の生成時、各プログラムが生成した頂点数はジオメトリシェーダの制限である 128 頂点以下であり、頂点数を制限内に抑えることを可能とした。

1.2　論文構成

本論文は全 4 章にて構成する。2 章にて提案手法について述べ、3 章では評価と分析について述べる。4 章にてまとめを述べる。

第 2 章

提案手法

本章では、本研究で提案するジオメトリシェーダ上での制限を考慮した樹木の枝の自動生成に関して述べていく。なお、本研究における座標系は右手系を前提とする。

2.1 樹木の枝の自動生成に関して

まず、本論文では枝の生成方法を述べるためにいくつかの定義をしておく。

第 1 の定義は軸についてである。地面を xz 平面と平行なものとし、樹木の幹が伸びる軸を y 軸、方向をプラス方向とする。

第 2 の定義は 1 回のジオメトリシェーダによって生成された枝についてである。今回、1 回のジオメトリシェーダによって 128 頂点内で生成された枝のことをセグメントと定義する。

本手法ではジオメトリシェーダに以下の 5 つの数値を渡すことによって枝を生成していく。

- 軸枝を作り始める 3 次元上の座標ベクトル $\mathbf{S}$
- 軸枝が伸びていく方向を表す方向ベクトル $\mathbf{A}$
- 軸枝の長さを表す実数 l
- 軸枝から生成する小枝の本数 n

- 小枝から生成する階層枝の最低階層を設定する数値 s

生成の順序は 3 つに分かれており、軸枝の生成、軸枝から小枝を生成、小枝の階層生成となっている。以上の生成の流れを以下の図 2.1 で示す。

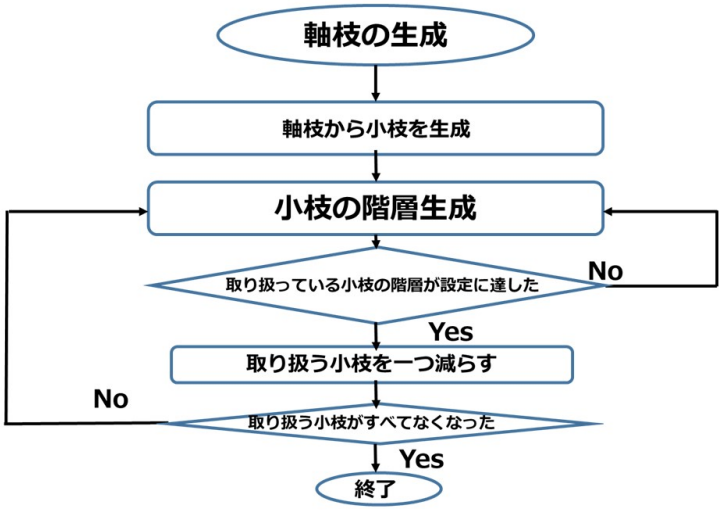

図 2.1　本手法の生成の流れ

2.2　軸枝の生成

この節ではセグメント生成の基準となる軸枝を生成を行う。また、この節で新しく生成する枝を軸枝と定義する。

まず、渡された数値のうち、座標ベクトル $\mathbf{S}$ と方向ベクトル $\mathbf{A}$ と長さを表す実数 l の 3 つを使う。渡された $\mathbf{S}$ の位置に軸枝の始点となる、新たな頂点 $\mathbf{B}_1$ を 1 つ生成する。上記の計算を示したものが式 (2.1) である。

$$\mathbf{B}_1 = \mathbf{S} \tag{2.1}$$

そして、$\mathbf{B}_1$ の方向ベクトルとして $\mathbf{C}_1$ を生成し、$\mathbf{A}$ を設定する。上記の計算を示したものが式 (2.2) である。

$$\mathbf{C}_1 = \mathbf{A} \tag{2.2}$$

$\mathbf{B}_1$ から $\mathbf{C}_1$ の方向へ長さ l 分進み、進んだ位置に新たな頂点 $\mathbf{B}_2$ を生成する。上記の計算を示したものが式 (2.3) である。

$$\mathbf{B}_2 = \mathbf{B}_1 + \mathbf{C}_1 \cdot l \tag{2.3}$$

また、$\mathbf{C}_1$ と同じように $\mathbf{C}_2$ を生成し、上記の式 (2.2) のように、生成した $\mathbf{C}_2$ に $\mathbf{C}_1$ の方向ベクトルを設定する。上記の計算を示したものが式 (2.4) である。

$$\mathbf{C}_2 = \mathbf{C}_1 \tag{2.4}$$

生成した 2 つの頂点を結ぶ形で枝を 1 つ生成する。これで軸枝が出来上がり、ここで生成した軸枝を基準にして 2.3 節以降、他の枝を生成する。上記の処理によって以下に示す図 2.2 のような枝の生成が行える。

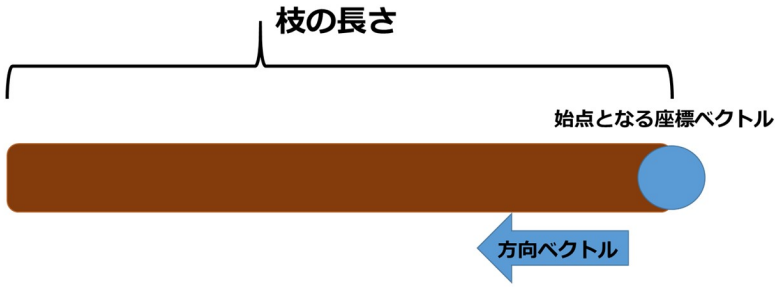

図 2.2　軸枝の生成

2.3　軸枝から小枝を生成

この工程では軸枝から小枝を渡された本数 n 分、生成していく。また、この節で新しく生成する枝を小枝と定義する。

引数として渡された、軸枝から生成する小枝の本数 n が 64 以上でなければ以下の処理を行う。まず初めに小枝を生成するために、本数 n の数だけ始点を生成する。

軸枝の根元から始点位置を 1 つずつ計算して決めていく。計算する始点位置 $\mathbf{D}$ と置いた場合、根元から順番に $\mathbf{D}_1$、$\mathbf{D}_2$ と計算していき、$\mathbf{D}_n$ まで計算を行う。始点位置の計算方法は、軸枝の終点 $\mathbf{B}_2$ を始点 $\mathbf{B}_1$ で引き、根本の 1 番近くに生成する始点を 1 として、何番目の始点であるかという番号をかける。その数値を n+1 で割り、$\mathbf{B}_1$ を足した数値が始点位置 $\mathbf{E}$ となる。上記の計算を示したものが式 (2.5) である。

$$\mathbf{D}_i = \frac{(\mathbf{B}_2 - \mathbf{B}_1) \cdot i}{n+1} + \mathbf{B}_1 \quad (i = 1, 2, \ldots n) \tag{2.5}$$

軸枝の根元から順番に、上記の計算で求めた位置に新しい頂点を生成する。

以上の処理により小枝の始点位置が決まる。図 2.3 は上記の始点位置の生成処理による実行結果を示したものである。

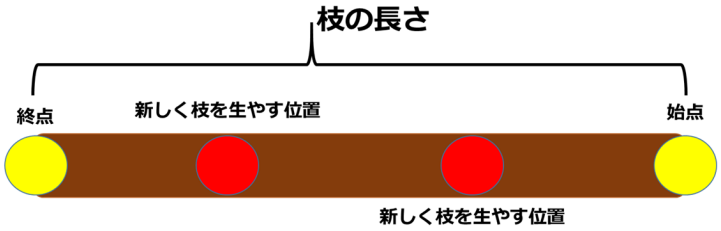

図 2.3　小枝を生成する始点位置の生成

次に小枝を生成するための終点を生成する。まず、枝を伸ばす長さを m、軸枝から生成される小枝の角度を α として設定する。生成する終点 $\mathbf{E}$ と置いた場合、対応した方向ベクトルを $\mathbf{F}_1$、$\mathbf{F}_2$ と生成していき、$\mathbf{F}_n$ まで生成する。軸枝の終点の方向ベクトル $\mathbf{C}_1$ に対し、y 軸中心の回転を行い設定しておいた角度 α だけ曲げる。式 (2.6) は y 軸を中心に回転する行列を示したもので

ある。

$$\mathbf{M}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \tag{2.6}$$

ただし、生成する小枝の本数が 2 本以上の場合、方向ベクトルの計算ごとに角度 α の符号を反転して方向ベクトルを曲げる。式 (2.7) は方向ベクトルの計算を表した式である。

$$\mathbf{F}_i = \begin{cases} \mathbf{M}_y(\alpha) \cdot \mathbf{C}_1 & (i \text{が奇数の時}) \\ \mathbf{M}_y(-\alpha) \cdot \mathbf{C}_1 & (i \text{が偶数の時}) \end{cases} (i = 1, 2, \ldots n) \tag{2.7}$$

上記の処理で生成した根元から順番に始点 $\mathbf{D}_1$、$\mathbf{D}_2 \cdots \mathbf{D}_n$ を元に終点 $\mathbf{E}$ の計算を行う。上記で計算した、各終点に対応する方向ベクトル $\mathbf{F}_i$ に一定の数値 m をかける。本研究では m の値を 10 とした。その数値に対応する始点位置を $\mathbf{D}_i$ を足したものが終点 $\mathbf{E}_i$ の位置となる。そして、計算をして求めた位置に終点となる頂点を生成する。終点の位置の計算を表したものが式 (2.8) である。

$$\begin{gathered} \mathbf{E}_i = \mathbf{D}_i + m \cdot \mathbf{F}_i \\ (i = 1, 2, \ldots n) \end{gathered} \tag{2.8}$$

生成する小枝の始点と終点になる頂点が生成できたので、その間に小枝を生成する。以下の図 2.4 は小枝の生成処理を示したものである。

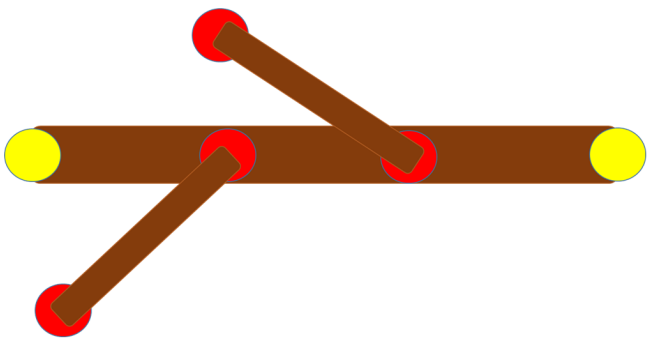

図 2.4　設定した位置から小枝を生成

最後に 2.4 節で繰り返し処理を行う回数として、変数 b を生成を行い数値を設定する。渡された s を確認し 1 以上なら、2.3 節で生成した小枝の本数 n を b に設定する。s が 0 である場合は n-2 を b に設定する。式 (2.9) は上記の計算を表した式である。

$$b = \begin{cases} n & (\text{if} \quad s \geqq 1) \\ n-2 & (\text{if} \quad s = 0) \end{cases} \tag{2.9}$$

2.4 小枝の階層生成

この節では、2.3 節で生成した小枝から階層枝を生成し、小枝の階層を作っていく。また、この節で新しく生成する枝を階層枝と定義する。小枝の階層とは、軸枝から生成された小枝を階層 0 とした場合、階層 0 の小枝から生成する 2 本の階層枝を階層 1、階層 1 の階層枝から生成する 2 本の階層枝を階層 2 とし、小枝を基準に何回目に生成された階層枝なのかという情報を小枝の階層と定義する。

この 2.4 節での処理の流れは、まず初めに小枝の階層が渡された数値 s になるまで 1 本の枝から階層枝を 2 本生成していく。その後、小枝の階層が s に達したら、根元に近い小枝から順番に階層枝の生成の処理をやめていく。最終的に生成される小枝の階層は、軸枝の根元に比べて先端の方が階層が多く、根元に一番近い小枝の階層は s、次の小枝の階層は s+1、次の小枝の階層は s+2 となり、軸枝の先端に向かうたび階層が 1 つ増えるように生成を行っていく。

まず、現在処理を行っている小枝の階層を d とし、初期値として 1 を設定する。

その後、2.2 節、2.3 節、2.4 節で生成した頂点の数を調べ、125 頂点に達していなければ以下の処理を行う。

現在の階層 d の 1 つ前の階層で作られた枝の始点と終点の情報を 1 つずつ取得し、その情報を元に階層枝を 2 本ずつ生成していく。まず、最初に 2 本の階層枝の生成のために始点位置の計算を行う。生成する階層枝の始点を $\mathbf{G}_1$、$\mathbf{G}_2$ とした場合、取得した始点を $\mathbf{I}_1$、$\mathbf{I}_2$ と置く。$\mathbf{I}_2$ から

$\mathbf{I}_1$ を引き、定数 0.3 をかけ、そのベクトルに $\mathbf{I}_1$ を足したものが、1 本目の始点位置である。上記の計算内でかけた定数 0.3 を 0.7 に変えたものが、2 本目の始点位置である。始点の位置の計算を表したものが式 (2.10) である。

$$\begin{aligned} \mathbf{G}_1 &= \mathbf{I}_1 + (\mathbf{I}_2 - \mathbf{I}_1) \cdot 0.3 \\ \mathbf{G}_2 &= \mathbf{I}_1 + (\mathbf{I}_2 - \mathbf{I}_1) \cdot 0.7 \end{aligned} \tag{2.10}$$

式 (2.10) で求めた $\mathbf{G}_1$、$\mathbf{G}_2$ それぞれの位置に新たに頂点を生成する。以下の図 2.5 は上記の処理によって生成される始点を示したものである。

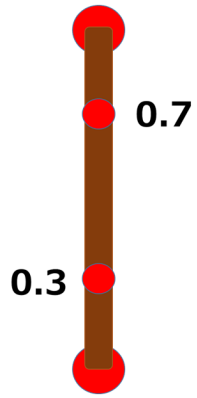

図 2.5　1 本の枝から 2 つの始点を生成

次に 2 本の階層枝の生成のために終点に対応する方向ベクトルの計算を行う。階層枝は、生成されるたびに伸びていく方向を変える。本研究では、その方向を変える回転処理を y 軸中心と z 軸中心に行う。それぞれ、y 軸中心に曲がる角度を角度 β、始点を生成した枝を軸として曲がる角度を γ、ω として設定しておく。

階層枝の終点に対応する方向ベクトルを $\mathbf{J}_1$、$\mathbf{J}_2$ とおき、その枝の方向ベクトルを $\mathbf{L}$ とする。式 (2.10) で使用した、1 つ前の階層の枝の終点 $\mathbf{I}_2$ に対応する方向ベクトル $\mathbf{L}$ に式 (2.6) を使い y 軸中心に回転を行う。1 本目に生成する階層枝の終点に対応する方向ベクトル $\mathbf{J}_1$ は、方向ベクトル $\mathbf{L}$ に対し、角度 β 分だけ回転を行う。2 本目に生成する階層枝の終点に対応する方向ベクトル

$\mathbf{J}_2$ は、方向ベクトル $\mathbf{L}$ に対し、角度 β の符号を反転して回転を行う。

その後 $\mathbf{L}$ を軸として中心に回転を行う。1 本目に生成する階層枝の終点に対応する方向ベクトル $\mathbf{J}_1$ は、角度 γ 分だけ回転を行う。2 本目に生成する階層枝の終点に対応する方向ベクトル $\mathbf{J}_2$ は、角度 ω 分だけ回転を行う。式 (2.11) は $\mathbf{L}$ を軸として回転する行列を示したものである。

$$\mathbf{M}_r(\theta) = \begin{pmatrix} \cos\theta + L_x^2(1-\cos\theta) & L_xL_y(1-\cos\theta) - L_z\sin\theta & L_zL_x(1-\cos\theta) + L_y\sin\theta \\ L_xL_y(1-\cos\theta) + L_z\sin\theta & \cos\theta + L_y^2(1-\cos\theta) & L_yL_z(1-\cos\theta) - L_x\sin\theta \\ L_zL_x(1-\cos\theta) - L_y\sin\theta & L_yL_z(1-\cos\theta) + L_x\sin\theta & \cos\theta + L_z^2(1-\cos\theta) \end{pmatrix} \tag{2.11}$$

式 (2.12) は上記の方向ベクトルの計算を表したものである。

$$\begin{aligned} \mathbf{J}_1 &= \mathbf{M}_r(\gamma) \cdot \mathbf{M}_y(\beta) \cdot \mathbf{L} \\ \mathbf{J}_2 &= \mathbf{M}_r(\omega) \cdot \mathbf{M}_y(-\beta) \cdot \mathbf{L} \end{aligned} \tag{2.12}$$

次に式 (2.12) で求めた終点の方向ベクトル $\mathbf{J}_1$、$\mathbf{J}_2$ を使い終点の位置を計算する。まず、階層枝 2 本の終点をそれぞれ $\mathbf{H}_1$、$\mathbf{H}_2$ とする。方向ベクトル $\mathbf{J}_1$、$\mathbf{J}_2$ に 2.3 節で作った数値 m をかけて、それぞれの始点位置 $\mathbf{G}_1$、$\mathbf{G}_2$ を足した位置が終点となる。そして、計算して求めた終点位置に新たな頂点を生成する。式 (2.13) は上記の計算を表したものである。

$$\begin{aligned} \mathbf{H}_1 &= \mathbf{G}_1 + m \cdot \mathbf{J}_1 \\ \mathbf{H}_2 &= \mathbf{G}_2 + m \cdot \mathbf{J}_2 \end{aligned} \tag{2.13}$$

生成する 2 本の始点と終点が決まったため、それぞれの 2 つの頂点の間に階層枝を生成する。

以下の図 2.6 は上記の処理内容を示したものである。

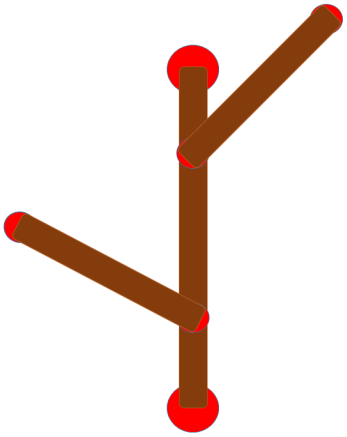

図 2.6　1 本の枝から階層枝を生成

上記の階層枝生成の処理を、頂点数の確認から b 回行うことで、階層が 1 つ前の枝すべてに階層枝を生成することができる。

b 回階層枝の生成処理を行った後、次の階層で階層枝を生成する本数を決める数値 f を設定する。今回生成した階層枝の本数 t とした場合、f の値の計算式は式 (2.14) のとおりである。

$$f = \begin{cases} t & (\text{if} \quad s > d-1) \\ t - 2^{d-1} & (\text{if} \quad s \leqq d-1) \end{cases} \tag{2.14}$$

f に設定する値が決まった後に d の値を 1 つ増やす。階層 d が 1 つ増えたため、新たな階層 d の階層枝の生成処理を頂点数の確認から繰り返し行う。

b の数が 0 以下である場合、生成する枝の形状が完成となるので上記の処理は終了する。生成した頂点数が 128 頂点に達した場合、強制的にすべての処理を終了する。

本手法のアルゴリズムによって生成したセグメントの実行結果が、以下の図 2.7 となっている。この工程を分割の数だけ行うことでジオメトリシェーダを使い樹木の枝を生成することができる。

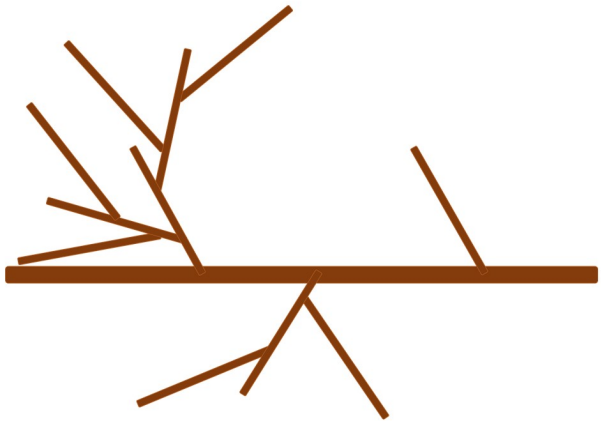

図 2.7　1 セグメントの生成例

2.5　セグメント生成の設定数値

2.1 節で 1 セグメントの生成に関する処理を述べた。しかし、渡される数値によっては、生成するセグメント同士が離れてしまい、バラバラな樹木の枝が出来上がってしまう。1.1 節で述べたように、本手法ではセグメント生成を 128 頂点内で行うため、樹木形状をいくつかのパーツに分けて生成する。そのため、ジオメトリシェーダに渡す数値は、樹木の幹またはいずれかのセグメントから繋がるものである必要がある。したがって、引数として渡す **S** はいずれかの軸枝の生成用に渡した **S** から **A** へ l だけ進む、線分上のいずれかの位置である必要がある。上記の条件にあった数値を与えた枝の生成例が以下の図 2.8 である。

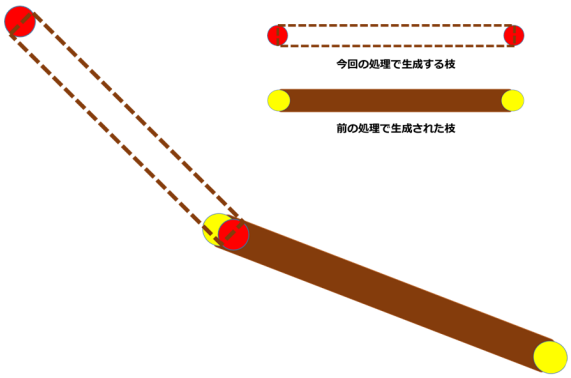

図 2.8　新たなセグメントを生成する位置決め

第 3 章

評価と分析

本章では、本手法によって生成された結果や評価について記述する。

3.1 本手法による生成結果

本手法の実験では、2 章で述べたセグメントの生成手法を C++ と FineKernelToolKitSystem[21] を用いて実装し、本手法の評価を行った。本手法を用いて、以下の数値を設定する。1 つ 1 つの処理を K_n と表記し、説明のため番号を K_0,K_1 と振っていく。番号は処理の識別をするためのものであり、順番を表すものではない。

- K_0 : $\mathbf{S}$ = (3.0, 3.0, 0.0), $\mathbf{A}$ = (1.0, 1.0, 0.0), $l = 10$, $n = 2$, $s = 0$
- K_1 : $\mathbf{S}$ = (6.0, 6.0, 0), $\mathbf{A}$ = (0.5, 0.3, 0.0), $l = 10$, $n = 2$, $s = 1$
- K_2 : $\mathbf{S}$ = (8.0, 8.0, 0.0), $\mathbf{A}$ = (0.7, 0.7, -0.4), $l = 10$, $n = 3$, $s = 1$

また、セグメントの生成時に行う回転の角度は α を 40 度、階層枝の生成時に行う回転の角度は β を 30 度、γ を-30 度、ω を 10 度に設定して行った。

以下に示してある図 3.1,3.2 は 1 セグメントに対して、上記で示した数値を元に本手法をそれぞれの K_n で生成したものである。

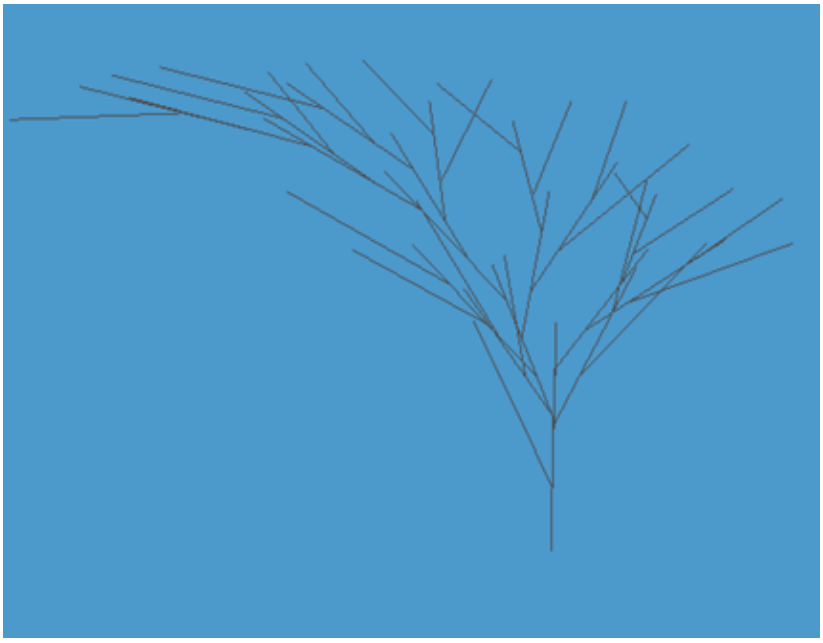

図 3.1　本手法の処理により生成したセグメントの実行結果正面 1

図 3.2　本手法の処理により生成したセグメントの実行結果側面 1

また、各 K_n の処理で生成されたセグメントに別々のマテリアル情報を与え、1 セグメントが枝のどこ部分なのか示したものが以下に示してある図 3.3,3.4 である。

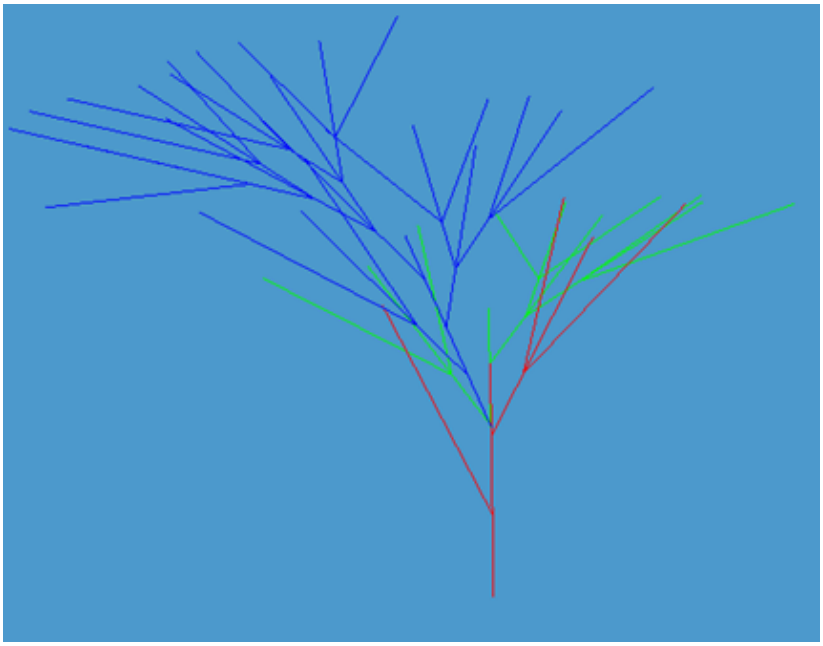

図 3.3　本手法の処理により生成したセグメントの実行結果正面 2

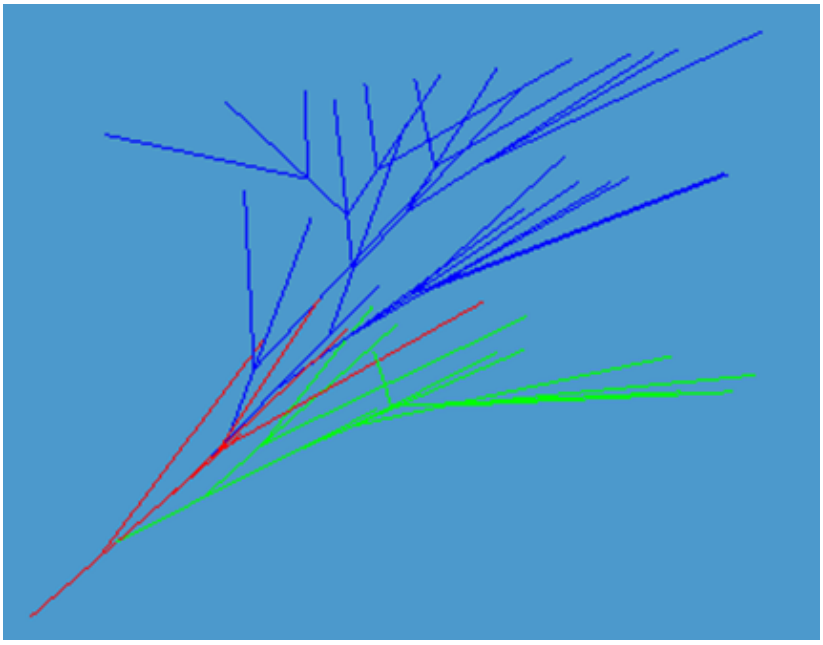

図 3.4　本手法の処理により生成したセグメントの実行結果側面 2

また、各処理でそれぞれ生成した頂点の数は以下の表 3.1 の通りである。

表 3.1　各処理で生成された頂点数

処理の番号	その処理で生成された頂点数
K_1	頂点数:10
K_2	頂点数:22
K_3	頂点数:108

任意の数値を与え、本手法を 3 回呼び出すことで枝の形状の生成に関して 128 頂点以下に抑えることができた。そのため本手法の目的としていた CPU 内でジオメトリシェーダの制限を考慮した樹木の枝の形状の自動生成を行うことができたといえる。

3.2　考察

本手法を用いることでジオメトリシェーダ上での制限内で樹木の枝の形状の自動生成を行うことができた。しかし以下の改善点が挙げられる。

- 自由な形状を生成することが可能であるが、条件分岐が存在するため処理が重い点
- 自己交差が起こってしまう点

第 1 の点は GPU の特徴である処理が速い点が、条件分岐を行うことで損なわれてしまうことである。本手法ではいくつかの処理で条件式を用いている。そのため高速な処理を行うことが難しい。特に 2.4 節での処理が最も条件式を用いているためこの部分の処理の条件式を減らすことにより改善が可能である。GPU の性能を十分に発揮するためには、今後枝の生成時に色々な場所で用いている条件分岐を極力使用しない工夫が必要である。

第 2 の点では、現在の手法では複雑な枝を生成した際に枝同士が衝突してしまう自己交差が起こってしまう点である。本来植物が成長していく過程で枝同士が衝突しそうな場合、衝突が起こらないように回避しながら成長が行われていく。そのため、自己交差が起こらないように処理をしていく必要となる。

今後の展望として上記に挙げた問題点を考慮してアルゴリズムの改善を行うことで、より良い樹木形状の自動生成が可能となるだろう。

第 4 章

まとめ

本研究では、GPU 内でジオメトリシェーダを使い樹木形状の生成を想定とした、CPU 内での樹木形状の生成アルゴリズムの提案を行った。ジオメトリシェーダの制限に対し、枝の自動生成を分割し、セグメント単位で生成を行うことで 128 頂点の制限内での生成を可能とした。また GPU が非同期で並列的に処理をするため逐次的な処理が難しい点では、軸となる枝を考慮し適切な数値を与えることで枝の生成を可能とした。したがって軸となる枝の情報を決めておけば、GPU を用いて自由に枝を生成することが可能である。以上のことからジオメトリシェーダを用いて制限内で樹木形状を生成でき、複雑な形状などの生成も行うことができる。

また 3 章でも述べたが以下のような改善点があげられる。

- 自由な形状を生成することが可能であるが、条件分岐が存在するため処理が重い点
- 自己交差が起こってしまう点

上記でも述べたが、本研究は GPU 内でジオメトリシェーダを使い樹木形状を生成することを想定し、CPU 内での樹木形状の生成アルゴリズムの提案を行った。しかし、将来的にはジオメトリシェーダを用いて樹木形状の自動生成を高速化したいと考えている。そのため 3.2 節で述べた改善点をしっかりと実装し、条件分岐による処理速度の低下が起きない自己交差を考慮した、樹

木形状の自動生成を実現させる必要がある。

謝辞

本研究についての相談、本論文の文章や数式に対する指導をしていただいた、渡辺先生、阿部先生に心より感謝いたします。

奇妙な日本語をクリスマス当日まで修正していただいた、院生のメンバーに感謝いたします。

研究や論文についての相談、大乱闘なストレス発散に協力してくれた、研究室のメンバーに感謝いたします。

英語の論文がうまく理解できない時に説明をしてくれた、留学生の友人に感謝いたします。

研究や論文のために家に帰らないことを承諾してくれた、両親に感謝いたします。

本論文の執筆に関わった全ての人たちに感謝いたします。

みんなありがとう。

参考文献

[1] Sony Interactive Entertainment Europe. HORIZON ZERO DAWN. https://www.jp.playstation.com/games/horizon-zero-dawn/. 参照:2018.01.16.

[2] SPIKE CHUNSOFT. WITCHER3. http://www.spike-chunsoft.co.jp/witcher3/. 参照:2018.01.16.

[3] M.Jaap. GPU-BASED PROCEDURAL PLACEMENT IN HORIZON ZERO DAWN. In *GPU-based Procedural Placement in Horizon Zero Dawn*, 2017.

[4] 竹内鋭典, 田中敏光, 佐川雄二, 杉江昇. 都市特有の条件を考慮した樹木の生長シミュレーション. 電気学会論文誌 C, Vol. 125, No. 12, pp. 1805–1811, 2005.

[5] 溝口敦士, 宮田一乗. 表面の成長による樹木の形状生成. 芸術科学会論文誌, Vol. 13, No. 1, pp. 45–58, 2014.

[6] 大志田憲, 村岡一信, 千葉則茂. 樹木の CG のための肥大生長シミュレーション. 情報処理学会研究報告グラフィクスと CAD(CG), Vol. 2000, No. 22, pp. 19–24, 2000.

[7] 溝口敦士. 階層構造を考慮した木の分布生成. 修士論文, Japan Advanced Institute of Science and Technology, 2008.

[8] SQUARE ENIX. FINALFANTASY15. http://www.jp.square-enix.com/ff15/. 参照:2018.12.24.

[9] 佐々木啓光, 村松瑞樹, 黒坂一隆. FINALFANTASY15 CHARACTE&ENVIRONMENT WORKFLOW . 2016.

[10] Hisa Ando. GPU を支える技術. 株式会社技術評論社, 2017.

[11] 株式会社フィックスターズ. OpenCL 入門　マルチコア CPU・GPU のための並列プログラミング. 株式会社インプレスコミュニケーションズ, 2010.

[12] David Wolff. OpenGl 4.0 シェーディング言語. 株式会社ボーンデジタル, 2012.

[13] Mark Segal and Kurt Akeley(松田晃一・内藤剛人・竹内俊治・神田崇史訳). OpenGl4.0 グラフィックスシステム. 株式会社カットシステム, 2010.

[14] The Industry's Foundation for High Performance Graphics. https://www.opengl.org/. 参照:2018.01.16.

[15] 赤木康宏, 北嶋克寛. GPU による形状生成に基づく樹木アニメーションの高速化. 情報処理学会研究報告グラフィックト CAD(CG), Vol. 109, No. 133, pp. 1–6, 2008.

[16] 柴原啓太, 佐藤尚. GPU を利用した物理シミュレーションに関する検討. 日本図学会, Vol. 41, No. 1, pp. 245–248, 2007.

[17] 奥屋武志, 坂井滋和. リアルタイムレンダリングにおける投影面上での曲がり具合いを考慮した輪郭線描画. 映像情報メディア学会技術報告, Vol. 40, No. 11, pp. 29–32, 2016.

[18] 奥野智江, 岡野紋, 加藤伸子, 狩野均, 西原清一. L システムを用いた道路網の生成. 情報処理学会, Vol. 56, No. 4, pp. 108–109, 1998.

[19] 大西克彦, 蓮池祥一, 北村喜文, 岸野文朗. インタラクティブな生長シミュレーションによる 3 次元樹木モデルの生成. 日本バーチャルリアリティ学会論文集, Vol. 11, No. 1, pp. 143–152, 2006.

[20] 進藤亜梨, 坂本雄児. 生長シミュレーションによるツタの CG モデル生成. 情報処理学会研究報告書, Vol. 2010-CG-141, No. 11, pp. 1–6, 2010.

[21] Fine Kernel Tool Kit System. https://gamescience.jp/FK/. 参照:2018.12.21.