Geração de vegetação com L-system

Ruan Chaves Rodrigues, Victor Alexandre de Carvalho Coelho e Vinícius Borges Alencar

Universidade Federal de Goiás

Computação Gráfica para Ciência da Computação

21/12/2018

SUMÁRIO

1 Introdução 02
2 Conceitos básicos 03
3 Proposta de solução 03
4 Experimentos 06
5 Conclusão 26
6 Referências bibliográficas 28

1. Introdução

Este trabalho foi desenvolvido para a matéria de Computação Gráfica ministrada pelo Professor Doutor Hugo Alexandre Dantas do Nascimento para a turma de Ciência de Computação da Universidade Federal de Goiás no segundo semestre de 2018.

O objetivo deste relatório é descrever os conceitos básicos das técnicas utilizadas para o desenvolvimento de um programa que implementa geração de vegetação seguindo um Sistema de Lindenmayer utilizando a ferramenta, o software e a linguagem de computação gráfica Processing [1].

O objetivo do programa é, seguindo o sistema inicialmente criado por Lindenmayer [2] em 1968 para a descrição de organismos multicelulares e, mais tarde melhorado para a descrição de plantas maiores e estruturas ramificadas completas, criar um programa que desenhe as estruturas descritas por uma gramática formal através do uso de reescrita paralela de símbolos. A técnica, chamada de L-systems, será descrita em seção posterior do trabalho.

O uso das funções nativas do Processing foram restringidas pelo professor, de modo que o sistema desenvolvido utiliza implementações realizadas pelos alunos em trabalhos anteriores ao final durante a disciplina. Assim, funções como as de translação, rotação e manipulação de objetos implementadas anteriormente foram utilizadas no sistema final.

2. Conceitos básicos

Um “Sistema de Lindenmayer” é um sistema de reescrita paralela com o auxílio de uma gramática formal. Foi inicialmente desenvolvido em 1968 pelo biólogo teórico húngaro Aristid Lindenmayer. O biólogo estudava estruturas multicelulares vegetais e desenvolveu um método de descrição formal do desenvolvimento e crescimento dessas estruturas. Mais tarde, o sistema de descrição foi estendido para estruturas mais complexas, como árvores e estruturas ramificadas completas.

Um L-system é composto por um alfabeto, com variáveis e/ou constantes, regras de reescrita e um axioma. Em seguida, o axioma inicial é reescrito recursivamente seguindo as regras de reescrita e substituição de modo que cada símbolo do alfabeto, seja ele uma constante ou uma variável, será uma instrução de representação no sistema gráfico ou de desenho no qual está sendo implementado.

A natureza recursiva do processo de reescrita paralela faz com que um L-system descreva facilmente estruturas fractais e autossimilares.

3. Proposta de solução

A interpretação de L-system para o programa descrito neste relatório resultou em alguns diferentes conjuntos de alfabetos, axiomas e regras para geração de árvores no sistema desenvolvido em Processing.

Foram escolhidos os seguintes conjuntos de L-systems 2D:

Espécie 0	Variáveis: **F** Constantes: **+**, **-**, **[**, **]** Axioma: **F** Regras: **F** -> **F[+F]F[-F]F**, **F** -> **F[-F]F[-F]F**, **F** -> **F[+F]F[-F]F**, **F** -> **F[-F]F[+F]F**
Espécie 1	Variáveis: **F** Constantes: **+**, **-**, **[**, **]** Axioma: **F** Regras: **F** -> **F[+F]F[-F][F]**, **F** -> **F[-F]F[-F][F]**, **F** -> **F[+F]F[-F][F]**, **F** -> **F[-F]F[+F][F]**
Espécie 2	Variáveis: **F** Constantes: **+**, **-**, **[**, **]** Axioma: **F** Regras: **F** -> **FF-[-F+F+F]+[+F-F-F]**, **F** -> **FF+[-F+F+F]+[+F-F-F]**
Espécie 3	Variáveis: **F**, **X** Constantes: **+**, **-**, **[**, **]** Axioma: **X** Regras: **F** -> **F[+X]F[-X]+X**, **F** -> **F[-X]F[-X]+X,** **F -> FF**
Espécie 4	Variáveis: **F**, **X** Constantes: **+**, **-**, **[**, **]** Axioma: **X** Regras: **F** -> **F[+X][-X]FX**, **F -> FF**
Espécie 5	Variáveis: **F**, **X**

	Constantes: **+**, **-**, **[**, **]** Axioma: **X** Regras: **F** -> **F-[[X]+X]+F[+FX]-X**, **F -> F-[[X]-X]+F[-FX]-X**, **F -> FF**
Espécie 6	Variáveis: **F** Constantes: **+**, **-**, **[**, **]** Axioma: **F** Regras: **F** -> **F[+F]F**, **F -> F[-F]F**

Para cada espécie, temos um conjunto de regras a serem seguidos para substituição de símbolos. Por exemplo, para uma planta a ser descrita pela espécie 6, temos a seguinte sequência de iteração.

Axioma: F
1ª recursão: F[+F]F, utilizando a primeira regra
2ª recursão: F[-F]F[+F[-F]F]F[-F]F, utilizando a segunda regra
3ª recursão: F[+F]F[-F[+F]F]F[+F]F[+F[+F]F[-F[+F]F]F[+F]F]F[+F]F[-F[+F]F]F[+F]F, utilizando a primeira regra

Cada símbolo do alfabeto representa uma ação a ser tomada para a estrutura a ser desenhada no programa. Segue abaixo a tabela de ações.

F	Avança um valor aleatório de acordo com a espécie.
X	Não realiza ação. Ajuda a manter controle da geração.

+	Realiza rotação em sentido horário. Graus definidos aleatoriamente de acordo com a espécie.
-	Realiza rotação em sentido anti horário. Graus definidos aleatoriamente de acordo com a espécie.
[	Guarda a posição atual em uma pilha.
]	Retorna a última posição guardada na pilha.

Os símbolos “**[**” e “**]**” servem para manter controle da altura da árvore, de modo que, como a geração é por níveis, é necessário guardar o nível das “bifurcações” para continuar o processo de geração e reescrita paralela. O desenho das estruturas ramificadas e árvores consiste na interpretação das strings de símbolos geradas por cada recursão da reescrita paralela utilizando o alfabeto e regras propostas para cada espécie.

4. Experimentos

Neste trabalho preferimos dividir o sistema em duas partes. Essa divisão nos permitiu uma distribuição de tarefas mais eficiente e um melhor desempenho para o sistema final.

A primeira parte é responsável pela geração das estruturas, que serão interpretadas como árvores, no formato de arquivos de texto com a extensão .txt. Nesses arquivos cada linha conterá cinco valores que representarão as coordenadas x, y e z, a distância máxima do ponto até uma folha e quantas ramificações existem no caminho do ponto até a raíz da árvore.

A segunda parte consiste na leitura desses arquivos .txt, definição de características para cada árvore e aplicação de transformações sobre o universo que conterá essas árvores projetadas.

Portanto, no final do trabalho foram produzidos dois programas. Um para a produção dos pontos das árvores e outro para leitura, interpretação e projeção desses pontos na tela.

A seguir será descrito a evolução do trabalho apresentando os problemas que surgiram e as decisões tomadas para solucionar cada um deles. Tais problemas remetem às duas partes do sistema e serão abordados em conjunto.

Problema 1

Estávamos aplicando as rotações no vetor que indicava a posição da tartaruga e não no vetor direcional da tartaruga. O que nos resultou nisso:

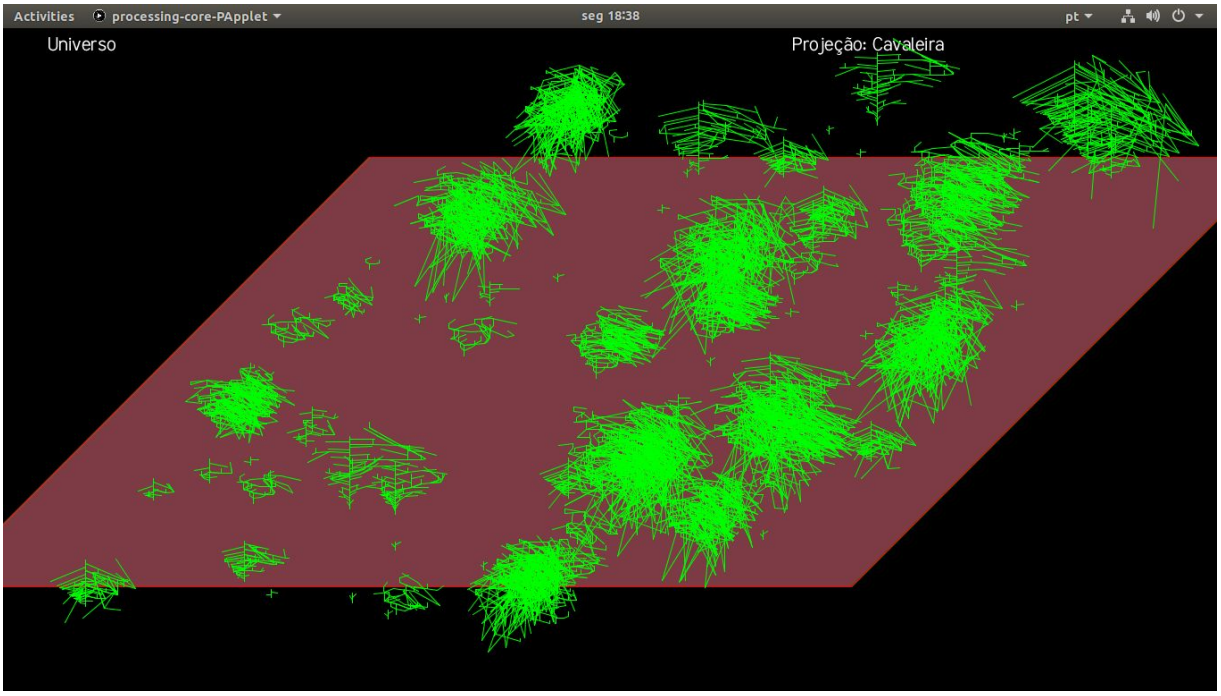


Imagem 1. Problema 1: Estruturas geradas ao não se diferenciar vetor direcional e vetor de posição.

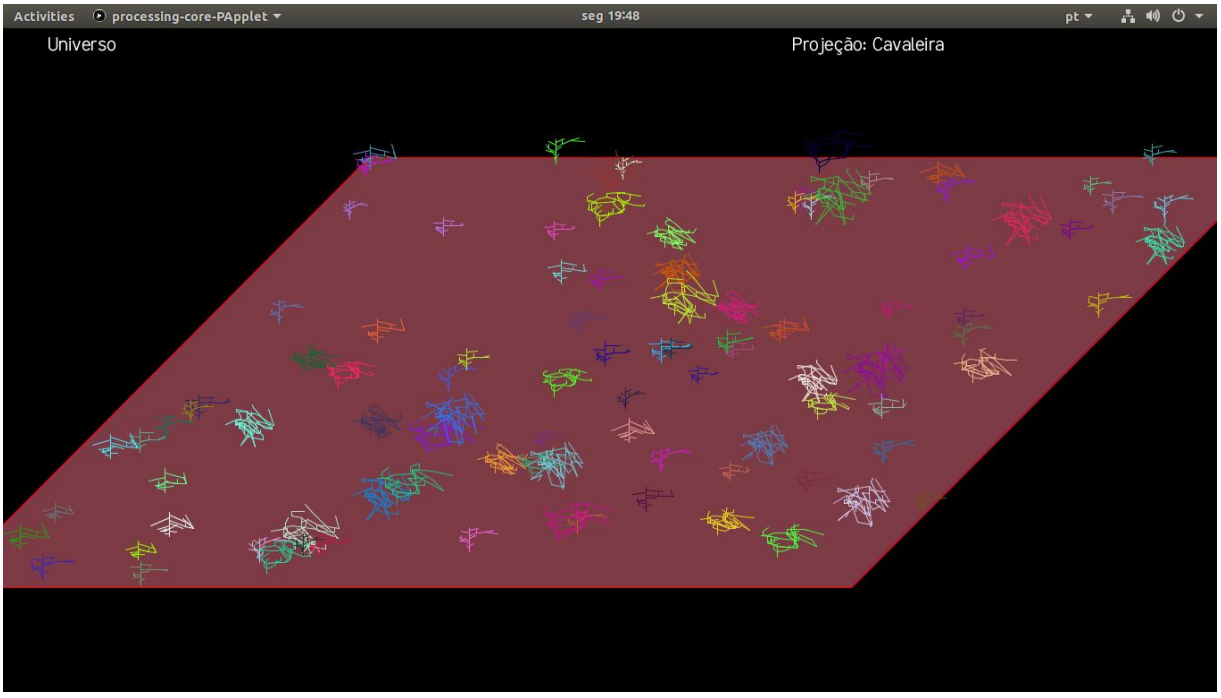


Imagem 2. Problema 1: Estruturas geradas ao não se diferenciar vetor direcional e vetor de posição (modelo colorido e com menos iterações).

Após a devida separação entre vetor direcional e vetor de posição o problema foi corrigido. E assim fomos capazes de gerar estruturas muito mais próximas de uma árvore:

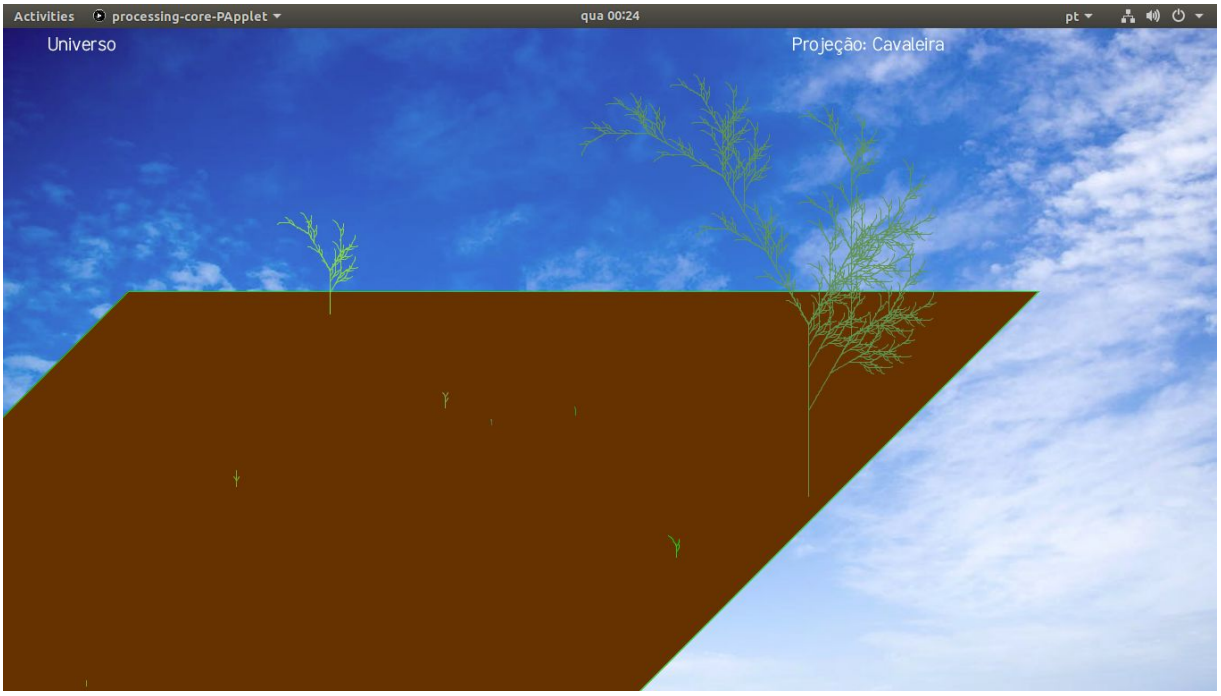


Imagem 3. Correção 1: Separação dos vetores de direção e posição.

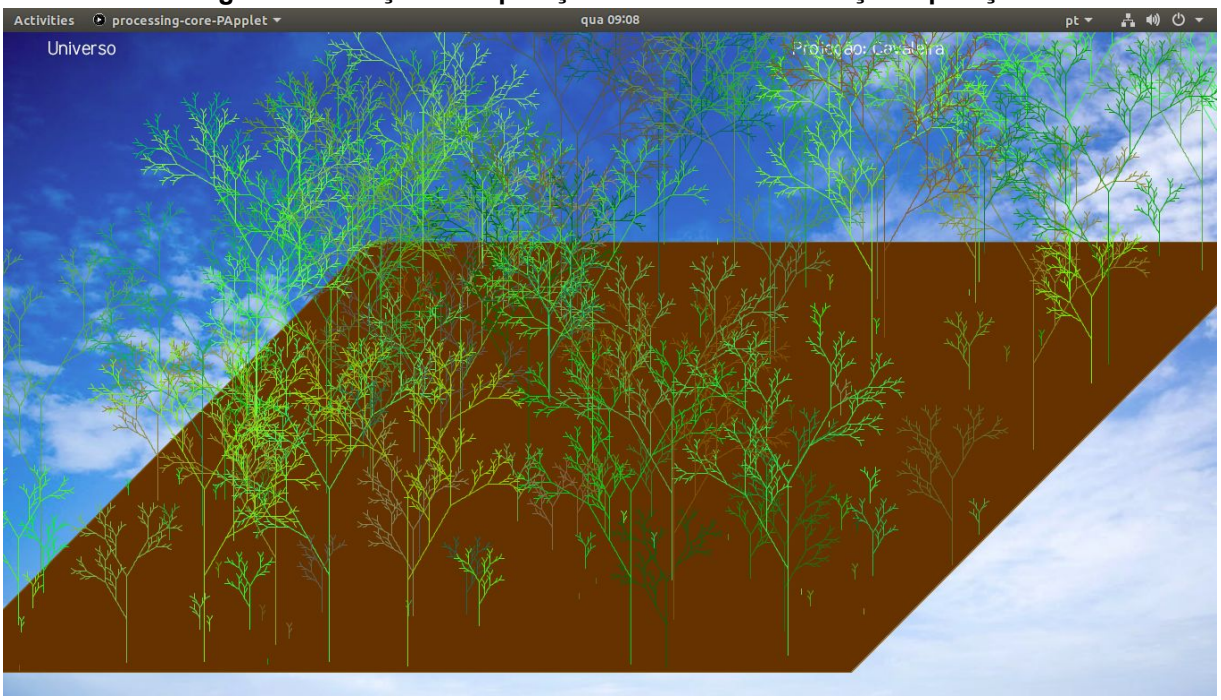


Imagem 4. Correção 1: Separação dos vetores de direção e posição (modelo com mais árvores).

Problema 2

Como visto na imagem acima, todas as árvores possuíam a mesma espessura para toda a estrutura. Foi assim que surgiu a necessidade de se saber em qual nível da estrutura um ponto pertencia. Para tal foi aplicado uma busca em largura ou BFS (*Breadth-First Search*). Com isso fomos capazes de obter o quinto valor do nosso arquivo .txt que nos indica o nível de cada ponto. Assim, poderíamos colocar uma espessura maior para pontos que estivessem mais próximos do solo, ou seja, pontos que fizessem parte do tronco principal e uma espessura menor para galhos e ramificações desses galhos.

Após a implementação dessa análise obtivemos o seguinte resultado:

Imagem 5. Correção 2: Uso do BFS para encontrar o nível de um ponto e definição da espessura de acordo com esse nível.

Problema 3

Após a correção do problema 2 surgiu um novo problema. Árvores pequenas estavam sendo geradas com um tronco muito espesso, totalmente desproporcional ao seu tamanho:

Imagem 6. Problema 3: Árvores pequenas com tronco muito espesso.

Para corrigir esse problema decidimos armazenar a altura máxima de cada árvore e com isso definimos a espessura de acordo não só com o nível do ponto, mas também de acordo com a altura máxima da árvore. Assim árvores pequenas teriam uma espessura menor do que uma árvore mais alta.

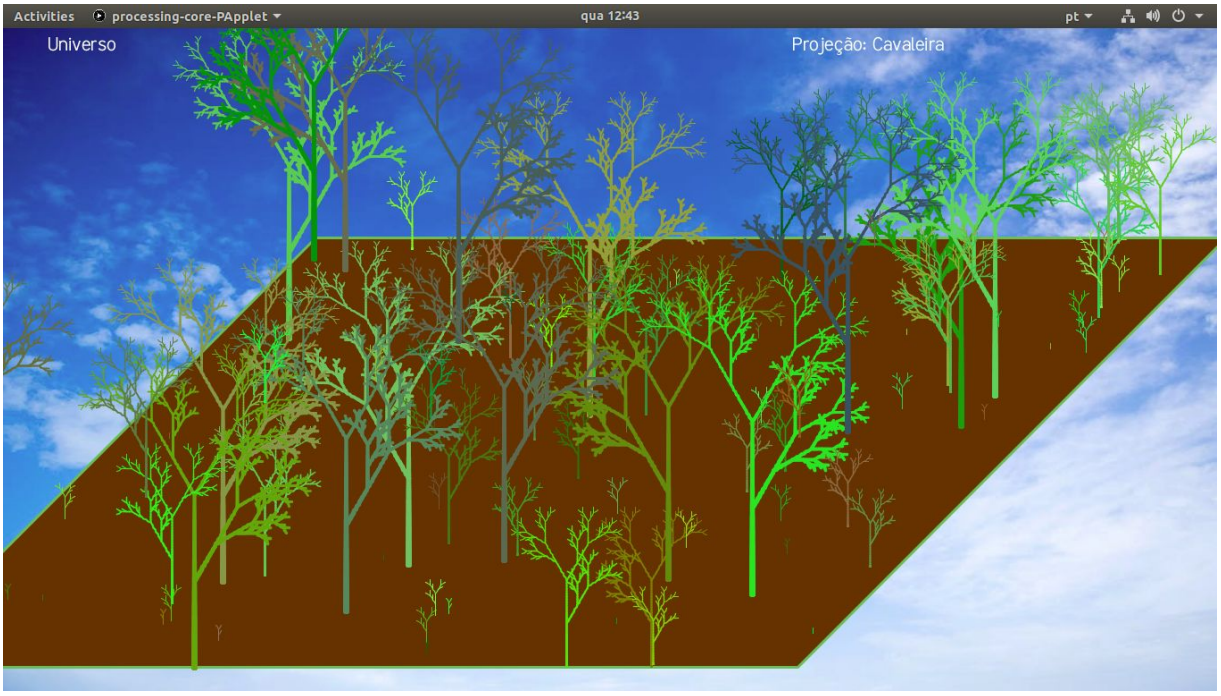


Imagem 7. Correção 3: Introdução da altura como parâmetro para definição da espessura.

Problema 4

A árvore não possuía folhas. Decidimos então recorrer novamente a busca em largura. Com ela obtivemos a distância máxima de um ponto até uma folha. Caso essa distância fosse 0, então o ponto em questão seria uma folha. Após identificar as folhas atribuímos uma coloração específica para elas:

Imagem 8. Correção 4: Uso do BFS para identificação de uma folha e atribuição de uma coloração específica para os pontos identificados como folhas.

Problema 5

A modelagem das folhas não parecia adequada. Pensamos então em modelar uma folha como um polígono específico com faces e aplicar uma textura nele. Entretanto isso se mostrou ser mais complexo do que o esperado, portanto decidimos, inicialmente, modelar a folha como um ponto de uma certa espessura:

Imagem 9. Tentativa de correção 5: Definição da folha como um ponto com uma certa espessura.

A figura vista de longe parece ter ficado boa, entretanto, ao se aplicar um zoom podemos perceber as falhas dessa tentativa de modelagem:

Imagem 10. Falha para a tentativa de correção 5: Definição da folha como um ponto com uma certa espessura.

Decidimos então ficar com o resultado obtido na imagem 8. Ou seja, os pontos identificados como folhas serão tratados como galhos normais, definidos por retas, mas receberão a coloração específica de uma folha e receberão uma espessura específica de uma folha.

Observação: Essa coloração específica foi definida por nós de acordo com o seguinte intervalo de cores Red{0, 178}, Green{51, 255} e Blue{0, 178}. Esse intervalo pode ser visualizado na figura abaixo:

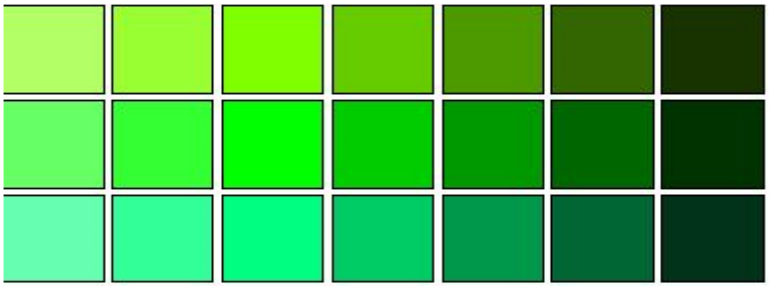


Imagem 11. Intervalo de Cores para as folhas

E a espessura da folha foi definida ao se aplicar a função do processing strokeWeight(3);

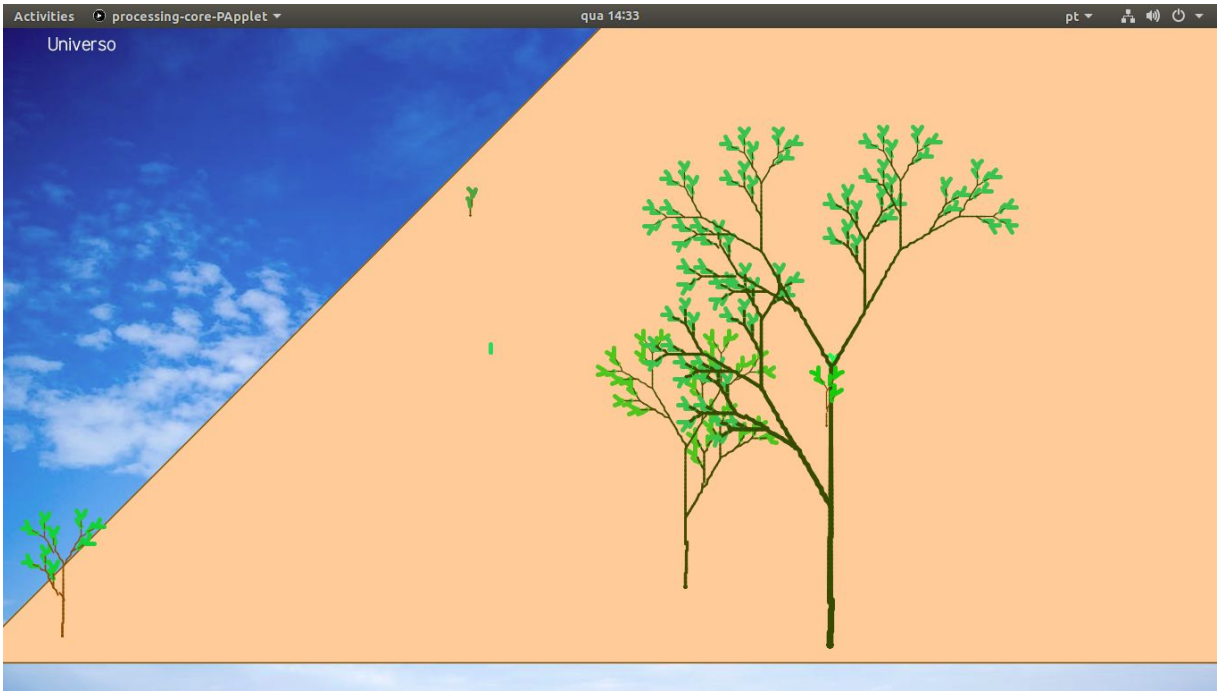


Imagem 12. Correção 5: Folhas modeladas como galhos que receberam coloração e espessura específicas.

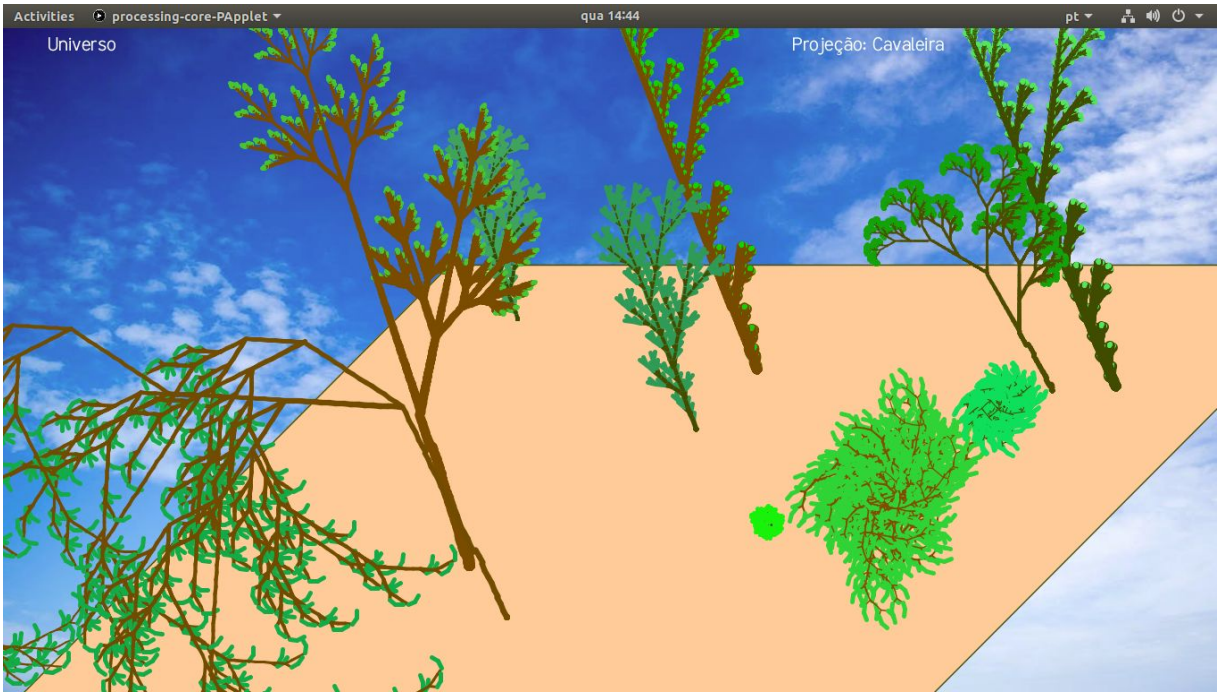


Imagem 13. Correção 5: Folhas modeladas como galhos que receberam coloração e espessura específicas.

Problema 6

As nossas árvores produzidas ainda não pareciam possuir as características mínimas de uma árvore real. Resolvemos então acrescentar flores. Para definir essas flores decidimos sortear aleatoriamente 20% das folhas e aplicar uma coloração própria de uma flor.

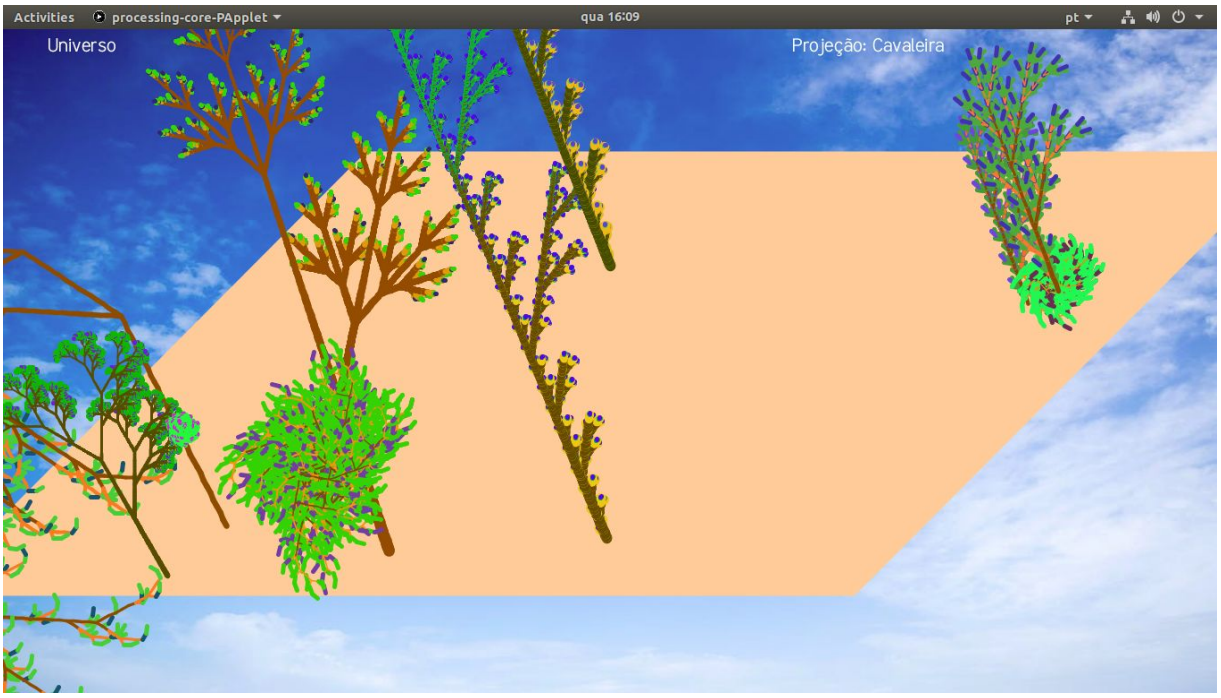


Imagem 14. Correção 6: Folhas recebem coloração específica de flores.

Observação: Nós definimos que toda árvore terá dois tipos flores, sendo que cada flor possuirá coloração distinta uma da outra. Como 20% das folhas serão sorteados para as flores, teremos que cada flor possuirá 10% de chance. Definimos também quatro grupos de flores possíveis para uma árvore. No momento de leitura do arquivo .txt, que contém a árvore, o programa, que projetará ela na tela, sorteará um grupo específico para cada uma das árvores lidas.

Grupo 1:

Flor1: Red {102, 255}, Green {0, 204} e Blue {0, 153}

Flor2: Red {0, 153}, Green {0, 255} e Blue {102, 255}

Grupo 2:

Flor1: Red {102, 255}, Green {0, 204} e Blue {0, 153}

Flor2: Red {51, 255}, Green {0, 153} e Blue {51, 255}

Grupo 3:

Flor1: Red {102, 255}, Green {102, 255} e Blue {0, 153}

Flor2: Red {0, 153}, Green {0, 255} e Blue {102, 255}

Grupo 4:

Flor1: Red {102, 255}, Green {102, 255} e Blue {0, 153}

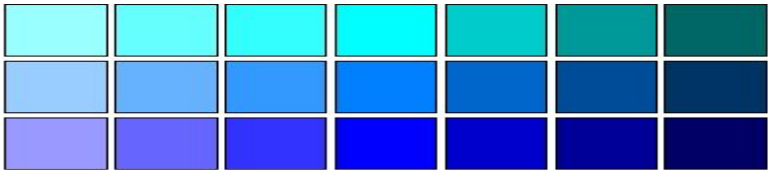
Flor2: Red {0, 153}, Green {0, 255} e Blue {102, 255}

As imagens abaixo representam os intervalos de cores acima.

Grupo 1:

Flor 1:

Flor 2:



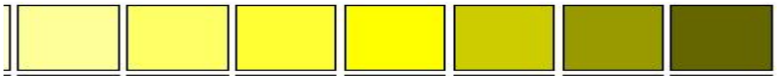
Grupo 2:

Flor 1:

Flor 2:

Grupo 3:

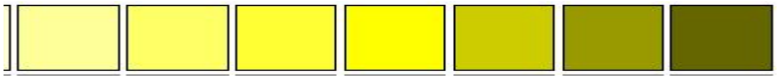
Flor 1:



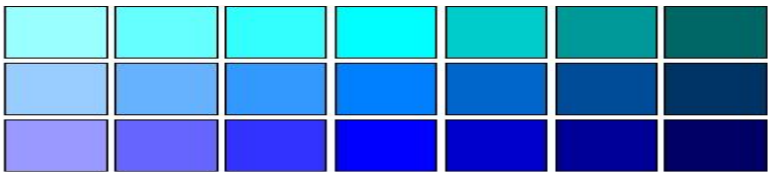
Flor 2:

Grupo 4:

Flor 1:



Flor 2:

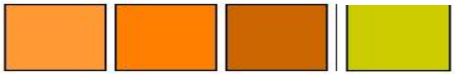


E no final decidimos usar o seguinte intervalo de cores para o tronco:

Tronco: Red {51, 153}, Green {25, 76} e Blue {0,0}

E o seguinte intervalo de cores para os galhos:

Galhos: Red {204, 255}, Green {102, 204} e Blue {0,51}



Resultado Final

Feito todas essas correções obtivemos o seguinte resultado:

Imagem 15. Resultado final com 150 árvores.

Imagem 16. Resultado final com 250 árvores.

Observações e detalhes finais

Por questões de desempenho, fizemos com que o programa de projetar as árvores fosse capaz de ler todos os arquivos da pasta de dados, mas que apresentasse na tela apenas uma parte dessas árvores. Na maioria dos testes a pasta de arquivos possuía 1000 árvores, mas apenas 100 delas eram plotadas na tela.

As árvores nunca terão uma raiz no mesmo ponto, pois no início do programa o chão é dividido, como se fosse uma malha, e então, cada ponto dessa malha será atribuído a uma única árvore. Essa atribuição é feita a partir da translação da árvore, criada inicialmente na origem, de acordo com os valores do ponto da malha.

É importante ressaltar que há uma ordenação das árvores, em ordem decrescente, usando os valores da coordenada Z como parâmetro. Isso garante que as árvores mais distantes do observador serão plotadas primeiro, evitando assim, que uma árvore mais próxima do observador seja sobreposta por outra mais distante. No programa é possível sortear um novo conjunto de árvores para serem plotadas e sempre que esse sorteio ocorrer haverá essa ordenação de acordo com a coordenada Z.

Observe que nesse trabalho foram usados modelos L-systems que geram estruturas 2D. Nesse tipo de estrutura as rotações são realizadas apenas em torno do eixo Z. Mas nós fizemos o uso de um “truque”, o qual possibilita a estrutura realizar uma rotação, de um ângulo fixo (no caso, 30 graus), em torno do eixo Y, toda vez que se abre uma nova ramificação. Esse truque gera a impressão da a árvore ser 3D. Esse efeito se torna explícito ao observar a rotação do universo nos vídeos abaixo.

Link do vídeo de uma floresta com 1000 árvores 2D sem o uso do “truque”:
https://drive.google.com/open?id=1yXrDJCXqAZ9xxZjxrTBBWBF_QJ0fRJpL

Link do vídeo de uma floresta com 1000 árvores 3D com o uso do “truque”:
https://drive.google.com/open?id=1m-mWs6NaVz-8MXC_pkfJ1xD3lp8qklQZ

Abaixo há uma imagem de cada uma das 7 espécies de árvores usadas na geração da floresta desse trabalho.

Imagem 17. Espécie 0.

Imagem 18. Espécie 1.

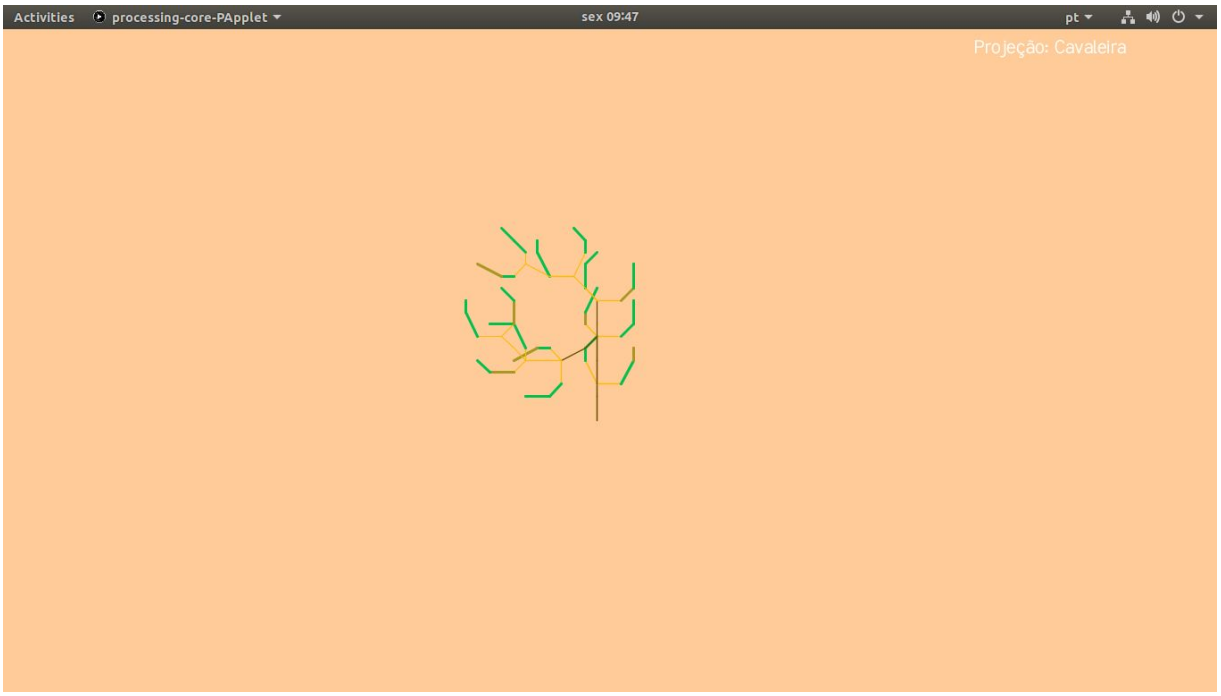


Imagem 19. Espécie 2.

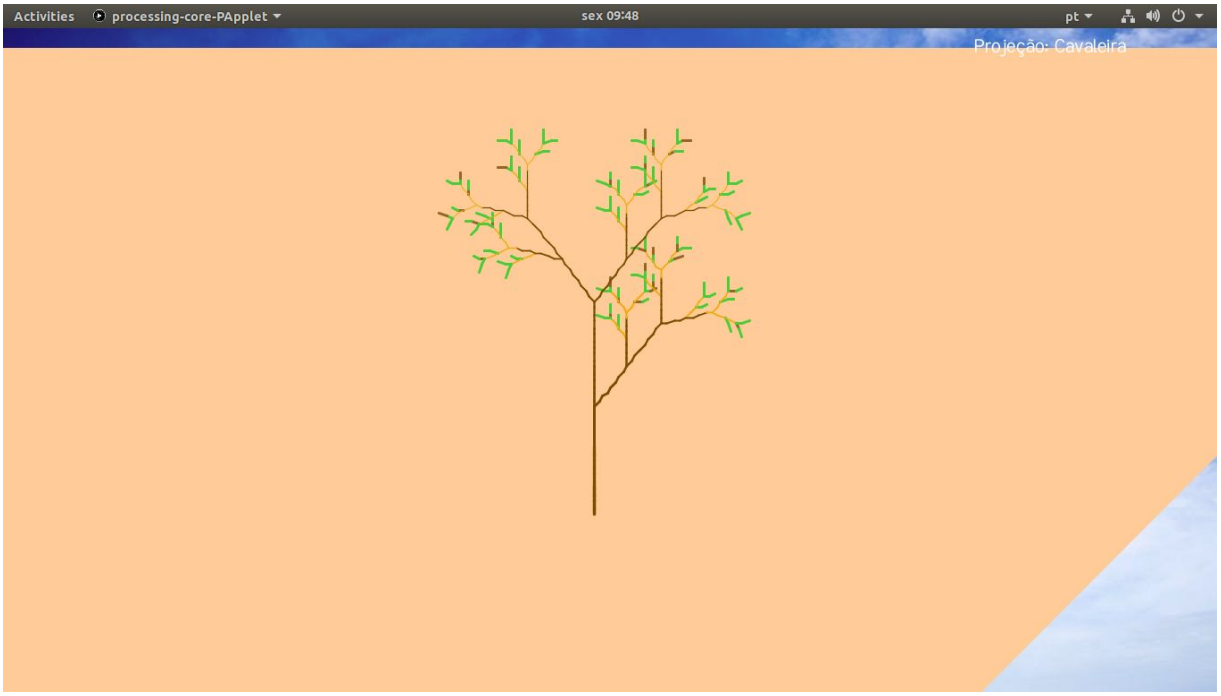


Imagem 20. Espécie 3.

Imagem 21. Espécie 4.

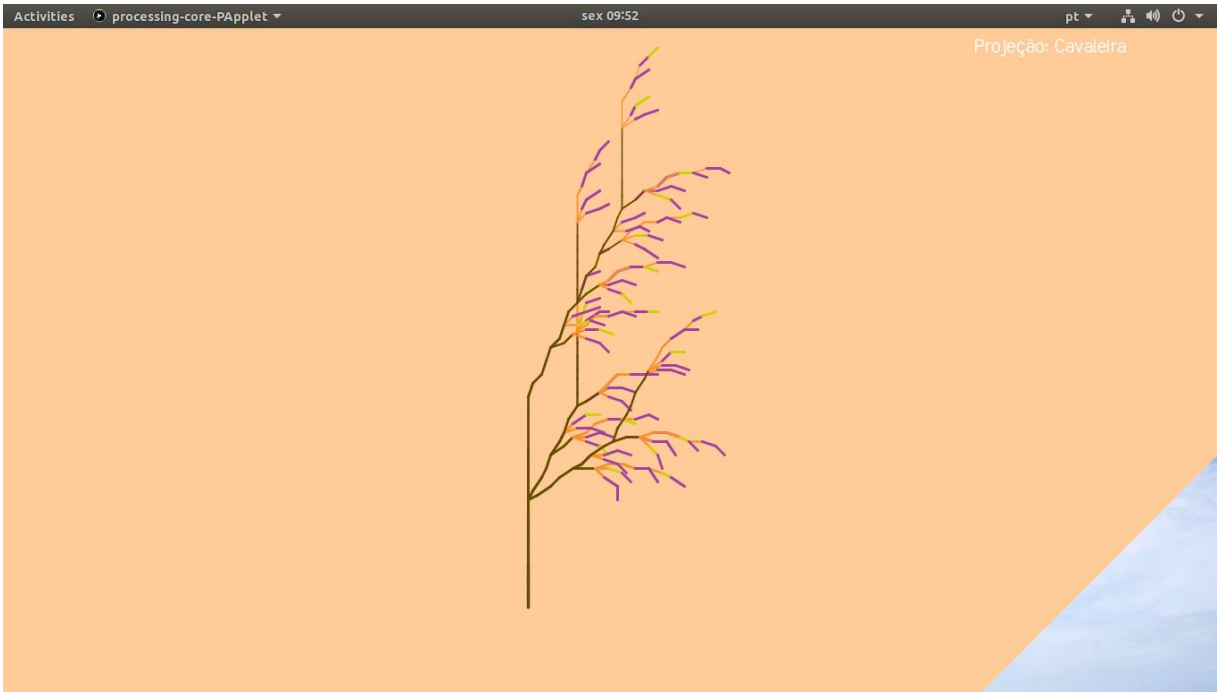


Imagem 22. Espécie 5.

Imagem 23. Espécie 6.

5. Conclusão

Neste trabalho foi abordado brevemente sobre os L-systems. Nessa abordagem foi apresentada uma introdução e definição de um L-system.

Posteriormente foram apresentados os modelos L-system escolhidos por nós para a geração das árvores usadas em nosso trabalho.

Também foram apresentados os experimentos, problemas e soluções que encontramos durante a produção deste trabalho.

Na geração das árvores, as principais dificuldades foram a modelagem da tartaruga, para que ela gerasse uma ilusão

tridimensional; e também o ajustamento dos parâmetros de cada árvore (quantidade de gerações, tamanho do passo, ângulo de inclinação) para que fossem gerados objetos esteticamente aceitáveis.

Na projeção das árvores na tela, a principal dificuldade foi transformar os pontos, gerados na etapa anterior, em um modelo que possuísse um mínimo de características para que ele pudesse ser classificado como uma árvore.

Uma boa sugestão de trabalho futuro seria adaptar este código para Android, como um papel de parede dinâmico. O papel de parede iria percorrer e rotacionar em torno de florestas geradas pela nossa aplicação de acordo com os movimentos do usuário enquanto utiliza o celular.

O nosso trabalho é adequado para telefones celulares devido à baixa quantidade de recursos computacionais que são necessários, tornando-lhe um papel de parede dinâmico ideal. E a extensão do Processing para Android permite rapidamente adaptar o nosso código para telefones celulares, com mínima necessidade de reescrita.

Também seria interessante um trabalho que explorasse as aplicações de outros sistemas de Lindemeyer que não são árvores, como a curva de Hilbert, a problemas de computação gráfica. Sabemos, por exemplo, que softwares como o Blender usam a curva de Hilbert para gerar arrays que armazenam imagens próximas no espaço em posições vizinhas, realizando assim um melhor uso da cache do processador e otimizando o tempo de execução.

Por fim, o objetivo inicial, de se criar um programa para geração de vegetação com o uso do L-systems, foi cumprido. Os artefatos gerados por esse trabalho são: este relatório, programa de geração das árvores,

que pode ser encontrado em https://github.com/ruanchaves/trees e pelo programa de projeção das árvores que poderá ser encontrado em https://github.com/victor-alexandre/Trabalho-Final-CG.

A realização desse trabalho se mostrou desafiadora, mas bastante produtiva. Aprendemos a usar o Processing, aprendemos a modelar estruturas complexas como árvores a partir dos L-systems, fomos capazes de desenvolver nossas próprias funções para translação, rotação, escala, projeção de objetos 2D e 3D. E além de tudo isso, a experiência mais significativa que tivemos foi ver que nossas propostas de soluções, para os problemas que surgiram no decorrer da elaboração do trabalho, nos deram resultados extremamente positivos.

6. Referências bibliográficas

[1] Processing https://processing.org/
Acesso em: 17/12/2018

[2] L-systems http://algorithmicbotany.org/papers/lsfp.pdf
Acesso em: 17/12/2018

[3] The Algorithmic Beauty of Plants
http://algorithmicbotany.org/papers/abop/abop.pdf
Acesso em 17/12/2018

[4] https://codepen.io/acme_incorporated/pen/LeXpQy

Acesso em 14/12/2018

[5] https://www.youtube.com/watch?v=f6ra024-ASY

Acesso em 14/12/2018

[6] https://github.com/shiffman/The-Nature-of-Code-Examples

Acesso em 14/12/2018

[7] https://manas.tech/projects/3d-trees-generator/

Acesso em 15/12/2018

[8] https://www.rapidtables.com/web/color/RGB_Color.html

Acesso em 20/12/2018

[9] https://sawyerwelden.com/lindenmayer-systems/

Acesso em 14/12/2018

[10] https://jobtalle.com/lindenmayer_systems.html

Acesso em 16/12/2018

[11] http://hardlikesoftware.com/projects/lsystem/lsystem.html

Acesso em 16/12/2018