Atividade 1

Criação de padrões complexos com uso de ferramentas primitivas

3 de dezembro de 2024

Contexto

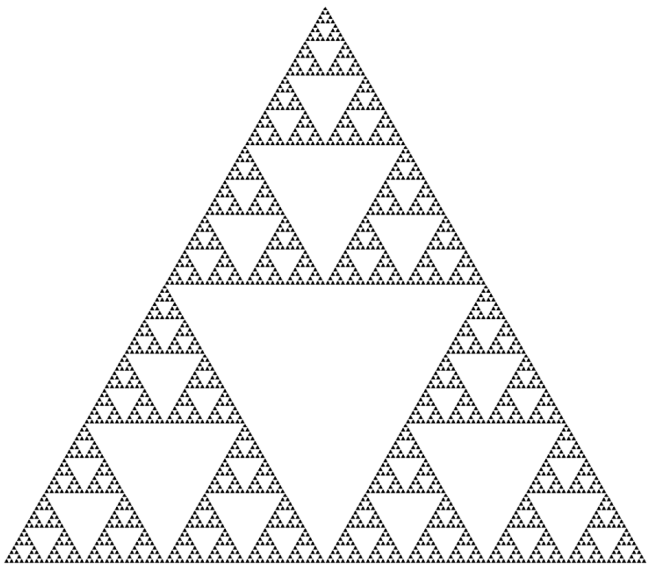
Fractais são estruturas matemáticas, com um padrão infinito. Em um fractal, existem cópias exatas do objeto inteiro em suas infinitas partes separadas. Algumas dessas estruturas são aleatórias, porém, outras podem ser definidas e geradas utilizando recursão.

Tais figuras recursivas são uma ótima maneira de exercitar e visualizar essa ideia de recursão, que geralmente é pouco compreendida de forma satisfatória. Pensando nisso, um de seus professores de Estrutura de Dados decidiu se aliar a outro professor, de Arq. e Org. de Computadores, para juntos construirem um programa utilizando uma das ferramentas mais baixo nível da programação: a linguagem Assembly.

Trabalhando em conjunto, os dois conseguiram construir uma boa parte do programa. Porém, ao chegar na parte recursiva, nenhum dos dois foi capaz de fazer avanços significativos. Determinados a não desistir, eles resolveram pedir ajuda para algum dos alunos do curso. E foi aí que você, confiante de suas habilidades adquiridas ao longo desses dois semestres, decidiu que seria o canditado ideal para terminar aquilo que nem os melhores conseguiram.

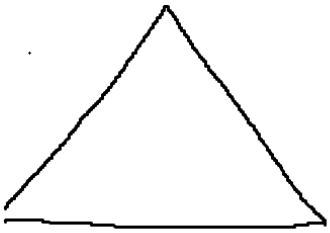
Problema

Na hora de escolher a figura que seria construida, a simplicidade foi levada em consideração, então eles optaram por desenhar um Triângulo de Sierpinski, uma das formas elementares da geometria fractal. Tal figura trata-se de um conjunto autossimilar de um triângulo. Se dividido em quatro outros triângulos congruentes entre si e entre o triângulo original, cujos vértices são os pontos médios do triângulo de origem, então os subconjuntos do fractal são três cópias escalonadas de triângulos derivados da iterada anterior.

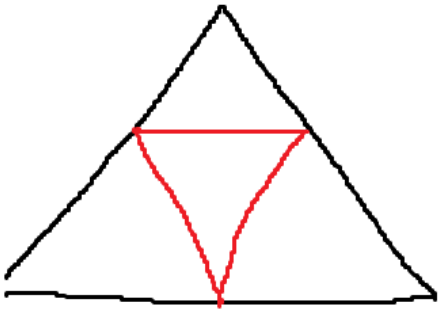


Para entender como a figura deve ser construida, os professores disponibilizaram as seguintes informações:

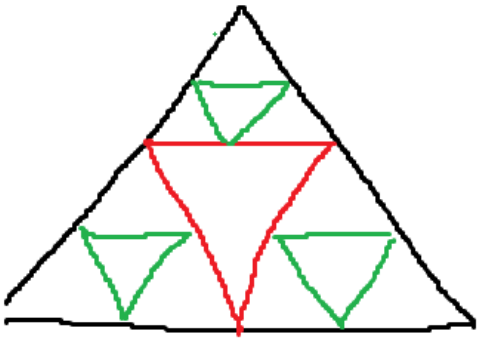
1) Começamos com um único triângulo. Esse é o único triângulo virado pra cima, todos os outros estarão ao contrário.



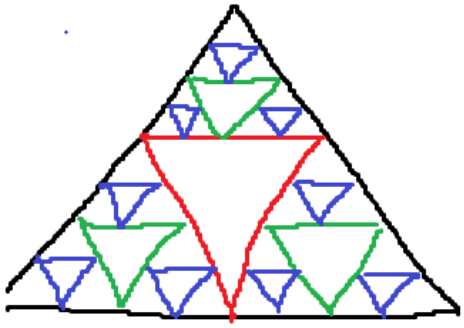
2) Dentro deste triângulo, desenhe um triângulo menor invertido. Seus cantos devem estar exatamente nos centros dos lados do primeiro triângulo.



3) Agora, desenhe 3 triângulos menores em cada um dos 3 triângulos que apontam para cima, novamente com os cantos nos centros dos lados dos triângulos que apontam para cima:



4) Agora existem 9 triângulos apontando para cima. Em cada um destes 9, desenhe novamente triângulos menores invertidos:



5) Repita o processo. Após passos infinitos, e se todos os triângulos apontando para cima forem preenchidos, você terá o Triângulo de Sierpinski.

Implementação

O código construido até o momento só faz o processo até o passo 2. Do passo 3 em diante, o processo se torna recursivo e cabe a você implementar.

Com o objetivo de tornar sua vida mais fácil, eles disponibilizaram o código base, para que você apenas adicione o que está faltando. Lembrando que, tudo que está no código base funciona e de forma alguma deve ser alterado.

Além disso, eles também decidiram fornecer uma implementação em C do algoritmo. Caso isso facilite seu processo de converter o código para instruções, o código está disponível a seguir:

```
//Declaration of the drawSierpinski function. The coordinates are the 3 outer corners of the Sierpinski Triangle.
void drawSierpinski(float x1, float y1, float x2, float y2, float x3, float y3);
//Declaration of the subTriangle function, the coordinates are the 3 corners, and n is the number of recursions.
void subTriangle(int n, float x1, float y1, float x2, float y2, float x3, float y3);

//depth is the number of recursive steps
int depth = 7;

//The main function sets up the screen and then calls the drawSierpinski function
int main(int argc, char *argv[])
{
  screen(640, 480, 0, "Sierpinski Triangle");
  cls(RGB_White); //Make the background white
  drawSierpinski(10, h - 10, w - 10, h - 10, w / 2, 10); //Call the sierpinski function (works with any corners inside the screen)
  //After drawing the whole thing, redraw the screen and wait until the any key is pressed
  redraw();
  sleep();
  return(0);
}

//This function will draw only one triangle, the outer triangle (the only not upside down one), and then start the recursive function
void drawSierpinski(float x1, float y1, float x2, float y2, float x3, float y3)
{
    //Draw the 3 sides of the triangle as black lines
    drawLine(int(x1), int(y1), int(x2), int(y2), RGB_Black);
    drawLine(int(x1), int(y1), int(x3), int(y3), RGB_Black);
    drawLine(int(x2), int(y2), int(x3), int(y3), RGB_Black);

    //Call the recursive function that'll draw all the rest. The 3 corners of it are always the centers of sides, so they're averages
    subTriangle
    (
      1, //This represents the first recursion
      (x1 + x2) / 2, //x coordinate of first corner
      (y1 + y2) / 2, //y coordinate of first corner
      (x1 + x3) / 2, //x coordinate of second corner
      (y1 + y3) / 2, //y coordinate of second corner
      (x2 + x3) / 2, //x coordinate of third corner
      (y2 + y3) / 2  //y coordinate of third corner
    );
}
```

```
//The recursive function that'll draw all the upside down triangles
void subTriangle(int n, float x1, float y1, float x2, float y2, float x3, float y3)
{
  //Draw the 3 sides as black lines
  drawLine(int(x1), int(y1), int(x2), int(y2), RGB_Black);
  drawLine(int(x1), int(y1), int(x3), int(y3), RGB_Black);
  drawLine(int(x2), int(y2), int(x3), int(y3), RGB_Black);

  //Calls itself 3 times with new corners, but only if the current number of recursions is smaller than the maximum depth
  if(n < depth)
  {
    //Smaller triangle 1
    subTriangle
    (
      n+1, //Number of recursions for the next call increased with 1
      (x1 + x2) / 2 + (x2 - x3) / 2, //x coordinate of first corner
      (y1 + y2) / 2 + (y2 - y3) / 2, //y coordinate of first corner
      (x1 + x2) / 2 + (x1 - x3) / 2, //x coordinate of second corner
      (y1 + y2) / 2 + (y1 - y3) / 2, //y coordinate of second corner
      (x1 + x2) / 2, //x coordinate of third corner
      (y1 + y2) / 2  //y coordinate of third corner
    );
    //Smaller triangle 2
    subTriangle
    (
      n+1, //Number of recursions for the next call increased with 1
      (x3 + x2) / 2 + (x2 - x1) / 2, //x coordinate of first corner
      (y3 + y2) / 2 + (y2 - y1) / 2, //y coordinate of first corner
      (x3 + x2) / 2 + (x3 - x1) / 2, //x coordinate of second corner
      (y3 + y2) / 2 + (y3 - y1) / 2, //y coordinate of second corner
      (x3 + x2) / 2, //x coordinate of third corner
      (y3 + y2) / 2  //y coordinate of third corner
    );
    //Smaller triangle 3
    subTriangle
    (
      n+1, //Number of recursions for the next call increased with 1
      (x1 + x3) / 2 + (x3 - x2) / 2, //x coordinate of first corner
      (y1 + y3) / 2 + (y3 - y2) / 2, //y coordinate of first corner
      (x1 + x3) / 2 + (x1 - x2) / 2, //x coordinate of second corner
      (y1 + y3) / 2 + (y1 - y2) / 2, //y coordinate of second corner
      (x1 + x3) / 2, //x coordinate of third corner
      (y1 + y3) / 2  //y coordinate of third corner
    );
  }
}
```

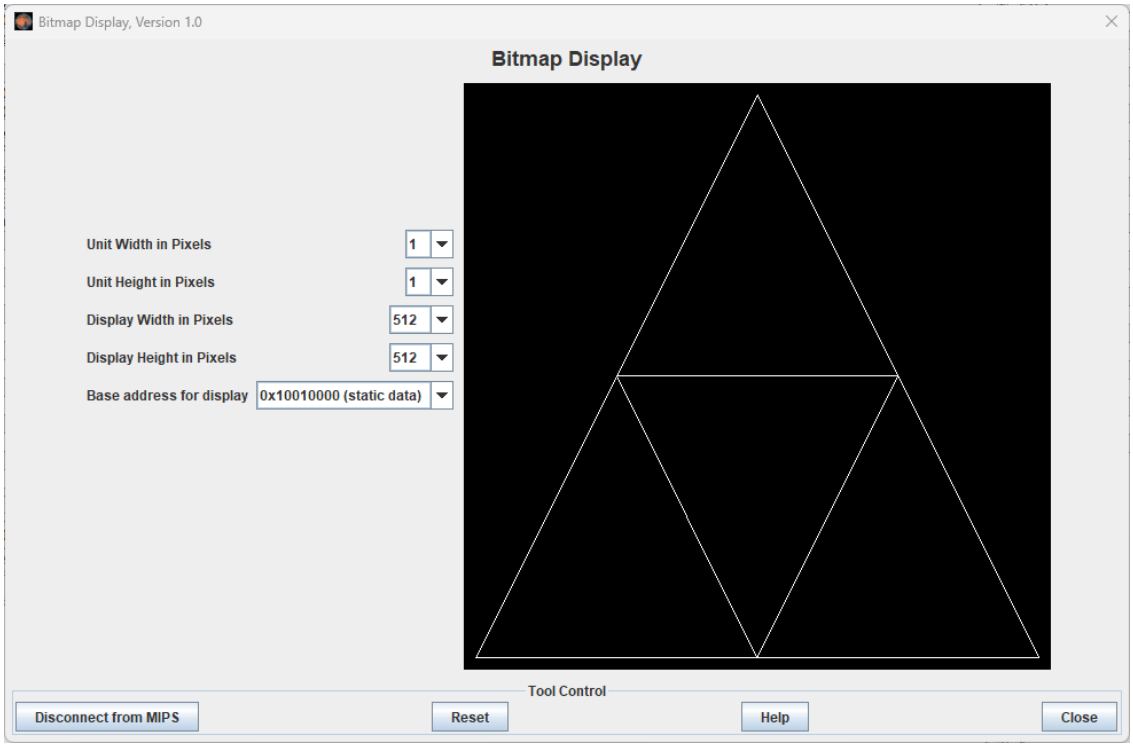
Informações adicionais

A função de desenhar linha em Assembly recebe dois pontos e traça uma linha entre eles, ela atua recebendo 4 argumentos: x1, y1, x2 e y2. Esses argumentos devem estar nos registradores $a0, $a1, $a2, e $a3, respectivamente.

A função de desenhar o sub triângulo recebe três pontos e desenha um triangulo para esses 3 pontos, além disso, deve receber a profundidade da recursão, que será usada para decidir o caso base. Tais argumentos são: x1, y1, x2, y2, x3, y3 e depth. Os 4 primeiros argumentos devem estar nos registradores $a0, $a1, $a2, $a3. Os três registradores restantes devem ser passados para a função através do uso da pilha.

Alguns macros são fornecidos para facilitar seu trabalho durante a recursão. Para salvar todos os registradores utilize o macro **save_recursion_registers** e para carregar de volta utilize o macro **load_recursion_registers**.

Para testar seu projeto, acesse a aba **Tools**>**Bitmap Display**. Configure o display para exibir 512 por 512. Após isso, se conecte ao mips e execute o programa. A saída esperada é essa:


Bitmap Display, Version 1.0
Bitmap Display
Unit Width in Pixels
1
Unit Height in Pixels
1
Display Width in Pixels
512
Display Height in Pixels
512
Base address for display
0x10010000 (static data)
Tool Control
Disconnect from MIPS
Reset
Help
Close