BỘ GIÁO DỤC VÀ ĐÀO TẠO
TR- ỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG
-------o0o-------

TÌM HIỂU PH- ƠNG PHÁP SINH ẢNH FRACTAL BẰNG HỆ HÀM LẶP (IFS) VÀ HỆ THỐNG L-SYSTEM

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Công Nghệ Thông Tin

Sinh viên thực hiện : Nguyễn Tam Hùng
Giáo viên h- ớng dẫn : PGS.TS Ngô Quốc Tạo
Mã số sinh viên : 101430

HẢI PHÒNG - 2010

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TR- ỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

-------o0o-------

ĐỒ ÁN TỐT NGHIỆP

NGÀNH CÔNG NGHỆ THÔNG TIN

HẢI PHÒNG 2010

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TR- ỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

-------*o0o*-------

NHIỆM VỤ THIẾT KẾ TỐT NGHIỆP

Sinh viên : Nguyễn Tam Hùng Mã số: 101430

Lớp : CT1001 Ngành: Công nghệ Thông tin

Tên đề tài:

Tìm hiểu ph- ơng pháp sinh ảnh Fractal bằng hệ hàm lặp (IFS) và hệ thống L-System

NHIỆM VỤ ĐỀ TÀI

1. Nội dung và các yêu cầu cần giải quyết trong nhiệm vụ đề tài tốt nghiệp

a. Nội dung:

b. Các yêu cầu cần giải quyết

2. Các số liệu cần thiết để thiết kế, tính toán

3. Địa điểm thực tập

PHẦN NHẬN XÉT ĐÁNH GIÁ CỦA CÁN BỘ CHẤM PHẢN BIỆN ĐỀ TÀI TỐT NGHIỆP

1. **Đánh giá chất l- ợng đề tài tốt nghiệp (về các mặt nh- cơ sở lý luận, thuyết minh ch- ơng trình, giá trị thực tế, ...)**

2. Cho điểm của cán bộ phản biện

(*Điểm ghi bằng số và chữ*)

.....

.....

Ngày.....tháng.....năm 2010

Cán bộ chấm phản biện

(Ký, ghi rõ họ tên)

LỜI CẢM ƠN

Trước hết, em xin chân thành cảm ơn thầy giáo PGS.TS Ngô Quốc Tạo đã tận tình hướng dẫn, chỉ dạy giúp đỡ tận tình và tạo mọi điều thuận lợi để em hoàn thành báo cáo tốt nghiệp của mình.

Em cũng xin chân thành cảm ơn trung tâm nghiên cứu và phát triển công nghệ phần mêm, nơi đã tạo điều kiện tốt trong suốt thời gian thực tập.

Em cũng xin chân thành cảm ơn quý thầy cô khoa công nghệ thông tin trường đại học dân lập Hải Phòng đã tận tình giảng dạy, trang bị cho chúng em những kiến thức cần thiết trong suốt quá trình học tập.

Và em cũng xin gởi lòng biết ơn đến gia đình, cha, mẹ,bạn bè đã ủng hộ, giúp đỡ và động viên em trong những lúc khó khăn.

Đề tài được thực hiện trong một thời gian tương đối ngắn, nên dù đã hết sức cố gắng hoàn thành đề tài nhưng chắc chắn sẽ không thể tránh khỏi những thiếu sót nhất định. Rất mong nhận được sự thông cảm và đóng góp những ý kiến vô cùng quý báu của các thầy cô, bạn bè, nhằm tạo tiền đề thuận lợi cho việc phát triển đề tài trong tương lai.

Hải Phòng, tháng 07 năm 2010

Sinh viên

Nguyễn Tam Hùng

LỜI NÓI ĐẦU

Tại sao môn hình học được xem là "khô cứng" và "lạnh lẽo"? Một trong lý do cơ bản nhất là vì nó không thể mô tả được thế giới tự nhiên xung quanh chúng ta. Những đám mây trôi lơ lững không phải là những quả cầu, những ngọn núi nhấp nhô không phải là những chóp nón, những bờ biển thơ mộng không phải là những đường tròn. Từ cảm nhận trực quan này, năm 1982, nhà toán học thiên tài Mandelbrot nảy sinh ra ý tưởng về sự tồn tại của một môn "Hình học của tự nhiên", Fractal Geometry. Từ đây, tôi và bạn có thể mô tả một đám mây một cách chính xác như một kiến trúc sư thiết kế căn nhà của họ.
Trong những năm gần đây, toán học và khoa học tự nhiên đã bước lên một bậc thềm mới, sự mở rộng và sáng tạo trong khoa học trở thành một cuộc thử nghiệm liên ngành. Cho đến nay nó đã đưa khoa học tiến những bước rất dài.Hình học Fractal bắt đầu được biết đến khoảng một thế kỷ nay,nó ra đời làm thay đổi cách nghĩ của chúng ta về khoa học nói chung và toán học nói riêng "Giới hạn của Fractal là không giới hạn". Nó mở ra một lãnh vực mới của toán học và có những tiến bộ đặc biệt trong vài chục năm gần đây, là môn khoa học trẻ được rất nhiều nhà khoa học quan tâm ở trên thế giới cũng như ở Việt Nam. Nhờ vào công cụ chính là máy tính có sự phát triển nhanh chóng về tốc độ mà việc biểu diễn Fractal trở nên dễ dàng hơn nên Fractal đã được đông đảo mọi người chú ý và thích thú nghiên cứu,các Fractal được sinh bởi máy tính có hình ảnh khá đẹp và rất thú vị. Với một người quan sát tình cờ màu sắc của các cấu trúc Fractal cơ sở và vẽ đẹp của chúng tạo nên một sự lôi cuốn hình thức hơn nhiều lần so với các đối tượng toán học đã từng được biết đến. Những nguyên nhân của sự lôi cuốn do hình học Fractal tạo ra là nó đã chỉnh sửa được khái niệm lỗi thời về thế giới thực thông qua tập hợp các bức tranh mạnh mẽ và duy nhất của nó.

Việc nghiên cứu ngôn ngữ hình học tự nhiên này mở ra nhiều hướng mới cho khoa học cơ bản và ứng dụng. Trong đề tài này chỉ mới thực hiện nghiên cứu một phần rất nhỏ về hình học phân hình và ứng dụng của nó. Nội dung của đề tài gồm có ba chương được trình bày như sau:

MỤC LỤC

CHƯƠNG I
TÌM HIỂU VỀ FRACTAL

1.1. SỰ HÌNH THÀNH VÀ PHÁT TRIỂN CỦA FRACTAL

"Khoa học hiện đại" vốn được phát triển từ kỷ nguyên Khai sáng (Enlightenment) ở thế kỷ 17, khởi đầu bởi những phát minh của Kepler, Galilei và Newton về các định luật của vận động vật chất và bởi sự thúc đẩy mạnh mẽ của cuộc cách mạng công nghiệp. Với những phát minh đó, lần đầu tiên con người tìm được một cách nhận thức thế giới bằng "phương pháp khoa học" mà không cần dựa vào một sức mạnh thần thánh nào hay phải viện đến những liên cảm huyền bí nào giữa trí tuệ con người với một tinh thần hay linh hồn của tự nhiên. Và cũng do đó, "khoa học" đã được phát triển trước hết và mạnh mẽ ở các lĩnh vực nghiên cứu tự nhiên như cơ học, vật lý học, thiên văn học, v.v... Cuối thế kỷ mười chín các nhà vật lý đã coi vật lý là lĩnh vực đóng kín và đầy đủ. Mọi sự việc, hiện tượng có tầm quan trọng trong lĩnh vực vật lý đều đã được phát hiện. Do đó chỉ cần làm rõ những sự kiện thiếu chặt chẽ sau đó đánh dấu "hoàn chỉnh". Vì thế trong khi sắp xếp một số sự kiện **không chặt chẽ** Schrodinger đã phát minh ra cơ học lượng tử và Einstein đã tạo ra thuyết tương đối. Do đó các nhà vật lý rất hoang mang vì xuất hiện nhiều vấn đề cần giải quyết hơn trước. Đối với Whitehead nhà toán học và triết học Mỹ thì một trong những khía cạnh có ý nghĩa nhất của tiến bộ này là quan điểm triết học về vật lý "tự nhiên không đến với ta sạch sẽ như ta nghĩ về nó", và khoa học, trong tinh thần qui giản của cơ giới luận, với việc làm sạch tự nhiên đó đã "hất đổ cả đứa bé cùng với chậu nước tắm" . Ta trở lại đối mặt với một tự nhiên và cuộc đời như nó vốn có, đầy cát bụi trần gian, lô nhô khúc khuỷu, gãy vỡ quanh co, chứ đâu có thẳng băng, tròn trịa như các hình vẽ của khoa học hình thức. Ta nhận ra điều đó cả từ trong chính bản thân phần cốt lõi tri thức của khoa học, cả từ những lĩnh vực ứng dụng khoa học đang có nhiều hứa hẹn thành công.

Nền tảng đầu tiên của Fractal đã được nhà toán học và vật lí học Leibniz đưa ra cùng khoảng thời gian đó là *self-similarity* (tính tự tương tự)

mặc dù chưa hoàn chỉnh nhưng đã mở ra bước tiến đầu tiên. Nhưng nó chỉ được biết đến với cái tên hình học Fractal đầu tiên vào năm 1872 khi Karl Weierstrass đưa ra một ví dụ với chức năng không trực quan của thuộc tính hiện thân khắp nơi liên tục mà không phụ thuộc vào không gian. Vào 1904, volt Helge Koch không hài lòng với kết luận của Weierstrass, đưa ra một định nghĩa hình học cao hơn về chức năng tương tự, mà bây giờ được gọi là *đường cong Koch*. Dựa trên thành quả đó , Waclaw Sierpinski đã xây dựng với tam giác vào năm 1915 mà sau nay gọi là *tam giác Sierpinski*. Ban đầu các Fractal hình học đã được mô tả như là những đường cong hơn là hình 2D mà ta được biết đến như là trong các công trình hiện đại ngày nay. Vào 1918, Bertrand Russell đã đoán nhận về một " vẻ đẹp tối cao " bên trong nẩy sinh trong toán học Fractal.Ý tưởng của các đường đồng dạng được cầm xa hơn nữa bởi Pierre Lévy Paul, người mà, trong 1938 đã đưa ra kiến giả về một đường cong fractal mới, *đường cong C Lévy*. Georg Cantor cũng đã cung cấp các ví dụ về các tập con cảu thuộc tính bất thường thực sự phù hợp – *tập Cantor* bây giờ cũng được công nhận là fractals. Những hàm lặp trong mặt phẳng phức được điều tra vào cuối thế kỉ 19 - đầu thế kỉ 20 bởi Henry Poincaré, Felix Klein, Pierre Fatou và Gaston Julian. Tuy nhiên, không có sự giúp đỡ của đồ họa máy tính hiện đại, họ thiếu những phương tiện để làm cho trực quan vẻ đẹp của nhiều đối tượng mà họ khám phá. Vào những năm 1960, Benoit Mandelbrot bắt đầu điều tra *self-similarity* (tính tự tương tự), mà trước đó được xây dựng trên công việc của Lewis Fry Richardson. Cuối cùng, vào 1975 Mandelbrot đưa ra từ "Fractal" để biểu thị một đối tượng mà có miền Hausdorff- Besicovitch là lớn hơn so với các miền trước đây. Ông ta minh họa định nghĩa toán học này bởi máy tính những trực quan hóa. Những ảnh này bắt đầu trở lên nổi tiếng dựa vào phép đệ quy, dẫn tới hình thành thuật ngữ "Fractal" ngày nay.

1.2. CÁC ỨNG DỤNG TỔNG QUÁT CỦA HÌNH HỌC FRACTAL

Hiện nay có 3 hướng ứng dụng lớn của lý thuyết hình học phân hình, bao gồm:

- Ứng dụng trong vấn đề tạo ảnh trên máy tính.
- Ứng dụng trong công nghệ nén ảnh.
- Ứng dụng trong nghiên cứu khoa học cơ bản.

□ ỨNG DỤNG TRONG VẤN ĐỀ TẠO ẢNH TRÊN MÁY TÍNH:

Cùng với sự phát triển vượt bậc của máy tính cá nhân trong những năm gần đây, công nghệ giải trí trên máy tính bao gồm các lĩnh vực như trò chơi, anmation video… nhanh chóng đạt đỉnh cao của nó. Công nghệ này đòi hỏi sự mô tả các hình ảnh của máy PC với sự phong phú về chi tiết và màu sắc với sự tốn kém rất lớn về thời gian và công sức. Gánh nặng đó hiện nay đã được giảm nhẹ đáng kể nhờ các mô tả đơn giản nhưng đầy đủ của lý thuyết fractal về các đối tượng tự nhiên. Với hình học phân hình khoa học máy tính có trong tay một công cụ mô tả tự nhiên vô cùng mạnh mẽ.

Ngoài các ứng dụng trong lĩnh vực giải trí, hình học phân hình còn có mặt trong các ứng dụng tạo ra các hệ đồ hoạ trên máy tính. Các hệ này cho phép người sử dụng tạo lập và chỉnh sửa hình ảnh, đồng thời cho phép tạo các hiệu ứng vẽ rất tự nhiên hết sức hoàn hảo và phong phú, ví dụ hệ phần mềm thương mại Fractal Design Painter của công ty Fractal Design. Hệ này cho phép xem các hình ảnh dưới dạng hình hoạ véctơ cũng như sử dụng các ảnh bitmap như các đối tượng. Như đã biết, các ảnh bitmap hiển thị hết sức nhanh chóng, thích hợp cho các ứng mang tính tốc độ, các ảnh véctơ mất nhiều thời gian hơn để trình bày trên màn hình (vì phải được tạo ra bằng cách vẽ lại) nhưng đòi hỏi rất ít vùng nhớ làm việc. Do đó ý tưởng kết hợp ưu điểm của hai loại đối tượng này sẽ giúp tiết kiệm nhiều thời gian cho người sử dụng các hệ phần mềm này trong việc tạo và hiển thị các ảnh có độ phức tạp cao.

□ ỨNG DỤNG TRONG CÔNG NGHỆ NÉN ẢNH:

Một trong những mục tiêu quan trọng hàng đầu của công nghệ xử lý hình ảnh hiện nay là sự thể hiện hình ảnh thế giới thực với đầy đủ tính phong phú và sống động trên máy tính. Vấn đề nan giải trong lĩnh vực này chủ yếu do yêu cầu về không gian lưu trữ thông tin vượt quá khả năng lưu trữ của các thiết bị thông thường. Có thể đơn cử một ví dụ đơn giản: 1 ảnh có chất lượng gần như chụp đòi hỏi vùng nhớ 24 bit cho 1 điểm ảnh, nên để hiện ảnh đó trên màn hình mày tính có độ phân giải tương đối cao như 1024x768 cần xấp xỉ 2.25Mb. Với các ảnh "thực" 24 bit này, để thể hiện được một hoạt cảnh trong thời gian 10 giây đòi hỏi xấp xỉ 700Mb dữ liệu, tức là bằng sức chứa của một đĩa CD-ROM. Như vậy khó có thể đưa công nghệ multimedia lên PC vì nó đòi hỏi một cơ sở dữ liệu ảnh và âm thanh khổng lồ.

Đứng trước bài toán này, khoa học máy tính đã giải quyết bằng những cải tiến vượt bậc cả về phần cứng lẫn phần mềm. Tất cả các cải tiến đó dựa trên ý tưởng nén thông tin hình ảnh trùng lặp. Tuy nhiên cho đến gần đây, các phương pháp nén thông tin hình ảnh đều có 1 trong 2 yếu điểm sau:

- Cho tỉ lệ nén không cao. Đây là trường hợp của các phương pháp nén không mất thông tin.
- Cho tỉ lệ nén tương đối cao nhưng chất lượng ảnh nén quá kém so với ảnh ban đầu. Đây là trường hợp của các phương pháp nén mất thông tin, ví dụ chuẩn nén JPEG.

Các nghiên cứu lý thuyết cho thấy để đạt một tỷ lệ nén hiệu quả (kích thước dữ liệu nén giảm so với ban đầu ít nhất hàng trăm lần), phương pháp nén mất thông tin là bắt buộc. Tuy nhiên một vấn đề đặt ra là làm thế nào có được một phương pháp nén kết hợp cả tính hiệu quả về tỷ lệ nén lẫn chất lượng ảnh so với ảnh ban đầu? Phương pháp nén ảnh phân hình được áp dụng gần đây bởi Iterated System đáp ứng được yêu cầu này.

Như đã biết, với một ánh xạ co trên một không gian metric đầy đủ, luôn tồn tại một điểm bất động x_r sao cho:

$$X_r = f(x_r)$$

Micheal F.Barnsley đã mở rộng kết quả này cho một họ các ánh xạ co f.Barnsley đã chứng minh được với một họ ánh xạ như vậy vẫn tồn tại một "điểm" bất động x_r.. Để ý rằng với một ánh xạ co, ta luôn tìm được điểm bất động của nó bằng cách lấy một giá trị khởi đầu rồi lặp lại nhiều lần ánh xạ đó trên các kết quả thu được ở mỗi lần lặp. Số lần lặp càng nhiều thì giá trị tìm được càng xấp xỉ chính xác giá trị của điểm bất động. Dựa vào nhận xét này, người ta đề nghị xem ảnh cần nén là "điểm bất động" của một họ ánh xạ co. Khi đó đối với mỗi ảnh chỉ cần lưu thông tin về họ ánh xạ thích hợp, điều này làm giảm đi rất nhiều dung lượng cần có để lưu trữ thông tin ảnh.

Việc tìm ra các ảnh co thích hợp đã được thực hiện tự động hoá nhờ quá trình fractal một ảnh số hoá do công ty Iterated System đưa ra với sự tối ưu về thời gian thực hiện. Kết quả nén cho bởi quá trình này rất cao, có thể đạt tỷ lệ 10000: 1 hoặc cao hơn. Một ứng dụng thương mại cụ thể của kỹ thuật nén phân hình là bộ bách khoa toàn thư multimedia với tên gọi "Microsoft Encarta" được đưa ra vào tháng 12/1992. Bộ bách khoa này bao gồm hơn 7 giờ âm thanh, 100 hoạt cảnh, 800 bản đồ màu cùng với 7000 ảnh chụp cây cối, hoa quả, con người, phong cảnh, động vật,… Tất cả được mã hoá dưới dạng các dữ liệu fractal và chỉ chiếm xấp xỉ 600Mb trên một đĩa compact.

Ngoài phương pháp nén phân hình của Barnsley, còn có một phương pháp khác cũng đang được phát triển. Phương pháp đó do F.H.Preston, A.F.Lehar, R.J.Stevens đưa ra dựa trên tính chất của đường cong Hilbert. Ý tưởng cơ sở của phương pháp là sự biến đổi thông tin n chiều về thông tin một chiều với sai số cực tiểu. Ảnh cần nén có thể xem là một đối tượng 3 chiều, trong đó hai chiều dùng để thể hiện vị trí điểm ảnh, chiều thứ ba thể hiện màu sắc của nó. Ảnh được quét theo thứ tự hình thành nên đường cong Hilbert chứ không theo hàng từ trái sang phải như thường lệ để đảm bảo các dữ liệu nén kế tiếp nhau đại diện cho các khối ảnh kế cạnh nhau về vị trí trong ảnh gốc. Trong quá trình quét như vậy, thông tin về màu sắc của mỗi điểm ảnh được ghi nhận lại. Kết quả cần nén sẽ được chuyển thành một tập tin có kích thước nhỏ hơn rất nhiều vì chỉ gồm các thông tin về màu sắc. Phương pháp này thích hợp cho các ảnh có khối cùng tông màu lớn cũng như các ảnh dithering.

□ ỨNG DỤNG TRONG KHOA HỌC CƠ BẢN:

Có thể nói cùng với lý thuyết topo, hình học phân hình đã cung cấp cho khoa học một công cụ khảo sát tự nhiên vô cùng mạnh mẽ như đã trình bày trong phần I.1, vật lý học và toán học thế kỷ XX đối đầu với sự xuất hiện của tính hỗn độn trong nhiều quá trình có tính quy luật của tự nhiên. Từ sự đối đầu đó, trong những thập niên tiếp theo đã hình thành một lý thuyết mới chuyên nghiên cứu về các hệ phi tuyến, gọi là lý thuyết hỗn độn. Sự khảo sát các bài toán phi tuyến đòi hỏi rất nhiều công sức trong việc tính toán và thể hiện các quan sát một cách trực quan, do đó sự phát triển của lý thuyết này bị hạn chế rất nhiều. Chỉ gần đây với sự ra đời của lý thuyết fractal và sự hỗ trợ đắt lực của máy tính, các nghiên cứu chi tiết về sự hỗn độn mới được đẩy mạnh. Vai

trò của hình học phân hình trong lĩnh vực này thể hiện một cách trực quan các cư xử kỳ dị của các tiến trình được khảo sát, qua đó tìm ra được các đặc trưng hoặc các cấu trúc tương tự nhau trong các ngành khoa học khác nhau. Hình học phân hình đã được áp dụng vào nghiên cứu lý thuyết từ tính, lý thuyết các phức chất trong hoá học, lý thuyết tái định chuẩn và phương trình Yang & Lee của vật lý, các nghiệm của các hệ phương trình phi tuyến được giải dựa trên phương pháp xấp xỉ liên tiếp của Newton trong giải tích số,… Các kết quả thu được giữ vai trò rất quan trọng trong các lĩnh vực tương ứng.

1.3. CÁC KIẾN THỨC TOÁN HỌC CƠ BẢN

1.3.1. Không gian Metric :

a) Không gian:

Định nghĩa 1:

Không gian X là một tập mà các điểm của không gian là các phần tử của tập đó.

Định nghĩa 2: (không gian Metric) :

Không gian (X, d) là một không gian metric nếu hàm $d:XxX\rightarrow R$ thoả mãn các điều kiện sau với d là khoảng cách giữa 2 điểm $x, y\in X$:

* $d(x, y) = d(y, x) \quad \forall x, y \in X$
* $0 < d(x, y) < \infty \quad \forall x, y \in X, x \neq y$
* $d(x, x) = 0 \quad \forall x \in X$
* $d(x, y) \leq d(x, z) + d(z, y) \quad \forall x, y, z \in X$

hàm d được gọi là metric .

Định nghĩa 3:

Hai metric d_1 và d_2 được gọi là tương đương trên X nếu tồn tại các hằng số $0<c_1, c_2<\infty$ sao cho:

$$c_1 d_1(x, y) \leq d_2(x, y) \leq c_2 d_1(x, y) \quad \forall (x, y)\in X\times X$$

Định nghĩa 4:

Hai không gian Metric (X_1, d_1) và (X_2, d_2) là tương đương nếu tồn tại hàm $h:X_1\rightarrow X_2$ là song ánh sao cho metric $\tilde{d}_1$ trên X_1 được định nghĩa bởi công thức:

$$\tilde{d}_1(x, y) = d_2(h(x), h(y)) \quad \forall (x, y)\in X_1$$

là tương đương với d_1.

Định nghĩa 5:

Hàm $f:X_1\rightarrow X_2$ từ không gian metric (X_1, d_1) và không gian metric (X_2, d_2) là hàm liên tục nếu với mỗi $\varepsilon >0$ và $x\in X_1$ $\exists\delta>0$ sao cho:

$$\mathbf{d_1(x, y)<\delta \Rightarrow d_2(f(x), f(y))<\varepsilon}$$

b) Dãy Cauchy, tập đóng, không gian metric đầy đủ:

Định nghĩa 1:

Dãy $\{x_n\}_{n=1}^{\infty}$ các điểm trên không gian metric (X, d) được gọi là dãy Cauchy nếu $\forall\varepsilon>0$, $\exists N$ là số tự nhiên sao cho:

$$d(x_n, x_m) < \varepsilon \quad \forall n, m>N$$

Định nghĩa 2:

Dãy $\{x_n\}_{n=1}^{\infty}$ các điểm của không gian metric (X, d) được gọi là hội tụ tới điểm $x\in X$ nếu với mọi $\varepsilon>0$, $\exists N$ là số tự nhiên sao cho:

$$d(x_n, x) < \varepsilon, \ \forall n<N$$

x được gọi là giới hạn của dãy và: $x = \lim_{n\to\infty} x_n$

Định lý:

Nếu dãy $\{x_n\}_{n=1}^{\infty}$ trong không gian metric (X, d) hội tụ tới điểm $x\in X$ thì dãy $\{x_n\}_{n=1}^{\infty}$ là dãy Cauchy.

Định nghĩa 3:

Không gian metric (X, d) là đầy đủ nếu mọi dãy Cauchy $\{x_n\}_{n=1}^{\infty}$ trong X có giới hạn $x\in X$.

Ví dụ: các không gian sau là không gian metric đầy đủ:

($\mathbb{R}$, d) ($\mathbb{R}^2$, d), với d là metric Ơclit.

Định nghĩa 4:

$S\subset X$ là tập con của không gian metric. Điểm $x\in X$ là điểm giới hạn của S nếu tồn tại dãy $\{x_n\}_{n=1}^{\infty}$ các điểm $x_n\in S\setminus\{x\}$ sao cho: $\text{Lim}_{n\to\infty}x_n=x$

Định nghĩa 5:

$S\subset X$ là tập con của không gian metric (X, d). Bao đóng của S (ký hiệu: $\bar{S}$) được định nghĩa như sau:

$\bar{S}= S \cup\{$các điểm giới hạn của S$\}$

S là tập đóng nếu nó chứa mọi điểm giới hạn của nó : $S= \bar{S}$

Ví dụ: $S = \{x=1/n; n=1, 2, ...\}$ là tập đóng trên ([0, 1], d), d là metric Ơcơlit.

c) Tập compact, tập giới nội , tập mở

Định nghĩa 1:

$S\subset X$ là tập con của không gian metric (X, d). Tập S là compact nếu mọi dãy vô hạn $\{x_n\}_{n=1}^{\infty}$ trong S đều chứa dãy con hội tụ trong S.

Định nghĩa 2:

$S\subset X$ là tập con của không gian metric (X, d). S là tập giới nội nếu tồn tại điểm $a\in X$ và số $R>0$ sao cho:

$$d(a, x)<R \qquad \forall x\in S$$

Định nghĩa 3:

$S\subset X$ là tập con của không gian metric (X, d). S là giới nội toàn phần nếu với mỗi $\varepsilon>0$ tồn tại tập hữu hạn các điểm $\{y_1, y_2, ..., y_n\}\subset S$ sao cho khi $x\in S$ thì

$$d(x, y_i) < \varepsilon \text{ với } y\in\{ y_1, y_2, ..., y_n\}$$

Định lý:

(X, d) là không gian metric đầy đủ $S\subset X$. S là tập compact và đầy đủ nếu nó đóng và giới nội toàn phần .

Định nghĩa 4:

$S \subset X$ là tập con của không gian metric (X, d). S được gọi là tập mở nếu với mỗi $s \in S$ $\exists \varepsilon > 0$ sao cho $B(x, \varepsilon) = \{y \in X : d(x, y) \leq \varepsilon\} \subset S$.

1.3.2. Không gian Hausdorff (H(X), h):

Phần này trình bày một số khái niệm về không gian Hausdorff là cơ sở để xây dựng fractal.

Định nghĩa 1:

(X, d) là không gian metric đầy đủ. Ký hiệu H(X) là tập các tập con compact của X.

Định nghĩa 2:

(X, d) là không gian metric đầy đủ , $x \in X$ và $B \in H(X)$. Khi đó khoảng cách từ điểm x tới tập B được xác định như sau:

$$d(x, B) = \text{Min}\{d(x, y) : y \in B\}.$$

Định nghĩa 3:

(X, d) là không gian metric đầy đủ A, $B \in H(X)$ khi đó khoảng cách từ tập A tới tập B được xác định như sau:

$$d(A, B) = \text{Max}\{d(x, B) : x \in A\}.$$

Định nghĩa 4:

(X, d) là không gian metric đầy đủ. Khoảng cách Hausdorff giữa các điểm A, $B \in H(X)$ được xác định như sau:

$$h(A, B) = d(A, B) \vee d(B, A)$$

Định lý:

h là metric trên H(X).

Chứng minh:

+) $h(A, B) = d(A, B) \vee d(B, A) = d(B, A) \vee d(A, B) = h(B, A)$

+) $A \neq B \in H(X) \Rightarrow$ có thể tìm được $a \in A, a \notin B : d(a, B) > 0$

$\Rightarrow h(A, B) \geq d(a, B) > 0$

+) $h(A, A) = d(A, A) \vee d(A, A) = d(A, A) = \text{Max}\{d(a, A) : a \in A\} = 0$

+) $d(a, B) = \min\{d(a, b) : b \in B\}, \quad a \in A$

$\leq \min\{d(a, c) + d(c, b) : b \in B\} \quad \forall c \in C$

$= d(a, C) + \min\{d(c, b) : b \in B\} \quad \forall c \in C$

$\leq d(a, C) + \max\{\min\{d(c, b) : b \in B\} : c \in C\}$

$\leq d(a, C) + d(c, B)$

$d(A, B) = \max\{d(a, B) : a \in A\} \leq d(a, C) + d(C, B)$

$\leq d(A, C) + d(C, B)$

tương tự có $d(B, A) \leq d(B, C) + d(C, A)$

$h(A, B) = d(A, B) \vee d(B, A)$

$\le (d(A, C)+d(C, B))\vee(d(B, C)+d(C, A))$

$\le d(A, C)\vee d(C, A)+d(C, B)\vee d(B, C)$

$\le h(A, C) + h(C, B)$

Định nghĩa 5:

$S\subset X$ và $\Gamma>0$ thì $S+\Gamma= \{y\in X : d(x, y) \le \Gamma$ với $x\in S\}$. Ta gọi $S+\Gamma$ được gọi là dao động của S bởi hình cầu bán kính Γ.

Bổ đề 1:

Cho $A, B \in H(X)$, (X, d) là không gian metric, $\varepsilon>0$. Khi đó ta có khẳng định: $h(A, B)\le\varepsilon \Leftrightarrow A\subset B+\varepsilon$ và $B\subset A+\varepsilon$.

Bổ đề 2: (bổ đề mở rộng)

(X, d) là không gian metric, $\{A_n : n = 1, 2, .., \infty\}$ là dãy Cauchy, các điểm trong (H(X), h), $\{n_j\}_{j=1}^{\infty}$ là dãy vô hạn các số nguyên $0<n_1<n_2<n_3<...$

Giả sử có dãy Cauchy $\{x_{nj} \in A_{nj} ; j=1, 2, ...\}$ trong (X, d) thì tồn tại dãy Cauchy $\{x_n\in A_n ; n\ge1\}$ sao cho

$$\tilde{x}_{nj} = x_{nj} \qquad \forall j = 1, 2, 3, ...$$

Định lý: (Về tính đầy đủ của không gian fractal)

(X, d) là không gian metric đầy đủ thì (H(X), h) cũng là không gian metric đầy đủ. Hơn nữa nếu $\{A_n\in H(X)\}_{n=1}^{\infty}$ là dãy Cauchy thì $A= \lim_{n\to\infty} A_n \in H(X)$. Có thể mô tả giới hạn của dãy như sau:

$$A= \{x\in X : \{x_n \in A_n\} \text{ là dãy Cauchy hội tụ đến } x\}$$

Định lý :

(X, d) là không gian metric, $\{x_n\}$ là dãy Cauchy hội tụ tới $x\in X$ ($\lim_{n\to\infty}d(x, x_n)=0$). nếu hàm $f : X \to X$ liên tục thì $\lim_{n\to\infty} f(x_n)=f(x)$.

1.3.3. Ánh xạ co

Định nghĩa 1:

Biến đổi $f:X\to X$ trên không gian metric (X, d) được gọi là co hay ánh xạ co nếu tồn tại hằng số $0\le s<1$ sao cho:

$$d(f(x), f(y)) \le s.d(x, y) \qquad \forall x, y\in X.$$

Khi đó s được gọi là hệ số co của f.

Định lý: (định lý ánh xạ co)

$f:X\to X$ là ánh xạ co trên không gian metric đầy đủ (X, d). Thì f có điểm cố định duy nhất $x_f\in X$, $\forall x\in X$ dãy $\{f^{on}(x) : n=0, 1, 2, ...\}$ hội tụ tới x_f tức là:

$$\lim_{n\to\infty} f^{on}(x) = x_f \qquad \text{đối với mỗi } x\in X .$$

Bổ đề 1:

Cho không gian metric (X, d) và ánh xạ w từ X lên chính nó. Nếu w:X→X là ánh xạ co trên (X, d) thì w liên tục.

Bổ đề 2:

w:X→X là ánh xạ liên tục trên không gian metric (X, d) thì w là ánh xạ từ H(X) vào H(X).

Bổ đề 3:

w:X→X là ánh xạ co trên không gian metric (X, d) với hệ số co s. Ta xác định biến đổi w:H(X)→H(X) như sau:

$w(B) = \{w(x): x \in B\} \quad \forall B \in H(X)$.

Khi đó w là ánh xạ co trên (H(X), h(d)) với hệ số co s.

Bổ đề 4:

Cho (X, d) là không gian metric.Nếu h là metric hausdorff thì

$$h(B \cup C, D \cup E) \leq h(B, D) \vee h(C, E) \qquad \forall B, C, D, E \in H(X)$$

Bổ đề 5:

(X, d) là không gian metric, $\{w_n: n=1, 2, .., N\}$ là các ánh xạ co trên (H(X), h) với hệ số co tương ứng của w_n là s_n. Ánh xạ W:H(X)→H(X) được xác định bởi:

$$W(B) = w_1(B) \cup w_2(B) ... \cup w_n(B) = \bigcup_{n=1}^{N} w_n(B)$$

với mỗi $B \in H(X)$ thì W là ánh xạ co với hệ số co $s=Max\{s_n: n=1, 2, .., N\}$.

1.3.4. Định lý cắt dán (COLLAGE)

Định nghĩa 1:

Cho (X, d) là không gian metric, biến đổi $w_0:H(X)\to H(X)$ được xác định $w_0(B)=C$ với mọi $B \in H(X)$. Thì w_0 được gọi là biến đổi cô đọng và C được gọi là tập cô đọng.

Định nghĩa 2:

Cho $\{X; w_n, n=1, 2, ..., N\}$ là hệ hàm lặp hyperbol với hệ số co $0 \leq s < 1$. Nếu $w_0:H(X)\to H(X)$ là một biến đổi cô đọng thì $\{X; w_n, n=0, 1, 2, ..., N\}$ là hệ hàm lặp hyperbol cô đọngvới hệ số co s.

Định lý:

Cho $\{X; w_n, n=0, 1, 2, ..., N\}$ là hệ hàm lặp hyperbol cô đọng với hệ số co s thì biến đổi W:H(X)→H(X) được xác định như sau:

$$W(B) = \bigcup_{n=0}^{N} w_n(B) \qquad \forall B \in H(X)$$

là ánh xạ co trên không gian metric đầy đủ (H(X), h(d)) với hệ số co s. Khi đó:

$$h(W(B), W(C)) \leq s.h(B, C) \qquad \forall B, C \in H(X)$$

điểm cố định duy nhất $A \in H(X)$ thoả mãn:

$$A = W(A) = \bigcup_{n=0}^{N} w_n(A) \text{ trong đó } A=\lim_{n\to\infty} W^{on}(B) \quad \text{với } B\in H(X).$$

Định lý(collage) :

Cho (X, d) là không gian metric đầy đủ.Cho tập $L\in H(X)$, và số $\varepsilon>0$. Chọn IFS$\{X;w_n, n=1, 2, ..., N\}$ (hoặc IFS cô đọng) với hệ số co $0\le s<1$ sao cho

$$h\left(L, \bigcup_{n=1(n=0)}^{N} w_n(L)\right) \le \varepsilon \qquad \text{với h(d) là metric Hausdorff}$$

thì

$h(L, A) \le \varepsilon/(1- s)$ với A là tập hút của IFS

$A=\lim_{n\to\infty} w_n(L)$

$A=L\cup w(L) \cup w^2(L) \cup ...=w^n(L)$

tức là :

$$h(L, A) \le (1-s)^{-1} h\left(L, \bigcup_{n=1(n=0)}^{N} w_n(L)\right) \qquad \forall L\in H(X).$$

Tập A và A_0 cho trước A, $A_0\in H(X)$, hệ $w_0, w_1, ..., w_n$ tạo thành hệ IFS hyperbol thì: $h(A, w(A_0)\cup w^2(A_0)\cup ...\cup ...)\le \varepsilon/(1-s)$ Nếu đầu tiên tập xuất phát A_0 khác tập cho trước A là ε thì sau lần lặp sẽ khác là $\varepsilon/(1-s)$. Ý nghĩa của định lý cắt dán nhằm đánh giá sự hội tụ của thuật toán lặp.

1.4. SỐ CHIỀU FRACTAL

Chúng ta hãy quay lại mệnh đề là đường cong mà số chiều Hausdorff-Besivitch lớn hơn số chiều Ơcơlit. Bây giờ chúng ta có một số ý tưởng về bản chất của đường cong fractal và định nghĩa mới về số chiều có nghĩa. Chúng ta không thể hiểu được fractal nếu như chúng ta đưa ra số đo có nghĩa. Định nghĩa nghiêm túc về số chiều Hausdorff-Besivitch là một tiến bộ đối với nhiều fractal hầu như không thể xác định được số chiều đối với phần lớn các fractal tự giống nhau. Giả sử rằng chúng ta bắt đầu với bộ khởi tạo dạng hình học đơn giản gồm một số đoạn thẳng liên thông. Nó có thể là tam giác, hình vuông hay thậm chí đường thẳng. Chúng ta hãy xác định một bộ tạo sinh gồm N đoạn thẳng, mỗi đoạn thẳng có độ dài r, ở đây r là một phần của đoạn thẳng đang được thay thế. Việc sắp xếp N đoạn thẳng sao cho khoảng cách từ tạo sinh tới điểm cuối của nó giống như độ dài của đoạn thẳng đang được thay trình thay thế lặp lại vô hạn lần và mỗi lần thay thế mỗi đoạn thẳng của trước bởi đoạn thẳng của bộ tạo sinh có co dãn tỉ lệ. Điều đó có thể thước Hausdorff-Besivitch của đường cong fractal là:

$$D= \log N/\log (1/r)$$

Việc so sánh số chiều này với số chiều Ơcơlit cho chúng ta một số tính chất của fractal. Chẳng hạn, D có giá trị 1.0 chỉ là đoạn thẳng thông thường, còn D có giá trị 2 có nghĩa là đường cong phủ hoàn toàn mặt phẳng.

2.1 CÁC THUẬT TOÁN DỰA VÀO HỆ HÀM LẶP

Michel Barnley, giáo sư toán học của trường đại học Công nghệ Atlanta Georgia đã nghiên cứu tập Julia, quan sát cách tạo ra và biến đổi có lẽ tạo sinh các dạng mà nó được đối sánh với các dạng của các sự vật trong trong thế giới xung quanh. Barnsley đã phát hiện ra hệ thống hàm lặp IFS(Iterated Function Systems).

Hệ thống như vậy bao gồm một tập các phương trình, mỗi phương trình biểu diễn phép quay, tịnh tiến và co dãn. Xuất phát từ một điểm và áp dụng một cách ngẫu nhiên các tập phương trình tương ứng với qui tắc xác suất được xác định. Barnsley có thể tạo ra fractal cổ truyền và phát hiện ra cách tạo ra cây dương xỉ và những hình dạng tự nhiên khác.

2.1.1. Thuật toán tiền định(Deterministic Algorithm)

Xét hệ hàm lặp IFS{ X;w_n, n = 1, 2, .., N}. Dựa trên ý tưởng tính toán trực tiếp dãy {A_n , n=1, 2, ...} từ tập ban đầu A_0

$A_1 = w_1(A_0) \cup w_2(A_0) \cup ..... \cup w_n(A_0)$

$A_2 = w_1(A_1) \cup w_2(A_1) \cup ..... \cup w_n(A_1)$

.....

$A_k = w_1(A_{k-1}) \cup w_2(A_{k-1}) \cup .... \cup w_n(A_{k-1}) = \cup_{i=1}^{n} w_i(A_{k-1}) \Rightarrow A_n = W^{on}(A)$

Dãy {A_n; n =1, 2, ...}$\subset$H(X) hội tụ tới tập hút của IFS trên metric Hausdorff.

$W = \lim_{n\to\infty} W^{0n}(A)$ là ảnh fractal cần tìm của hệ hàm lặp từ đó suy ra ứng dụng trong sinh ảnh fractal trên máy tính với số lần lặp đủ lớn ảnh thu được gần giống fractal cần tìm.

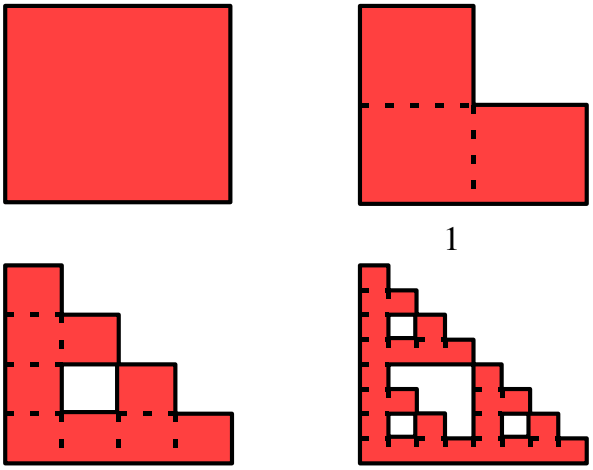
Để đơn giản ta xét IFS {R^2; w_n(n=1, 2, .., N)} với ánh xạ là các phép biến đổi A phin

$$w_i(x) = w_i\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} e_1 \\ f_2 \end{pmatrix} = A_i x + t_i$$

Ví dụ về tam giác Sierpinski, chúng ta có hệ hàm lặp sau:

$$w_1 = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad w_1 = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0 \end{pmatrix},$$

$$w_1 = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0 \end{pmatrix}$$



2.1.2. Thuật toán lặp ngẫu nhiên (Random Interation Algorithm):

IFS{X;w_1, w_2, ..., w_N}, w_i có hệ số co tương ứng s_i, xác xuất áp dụng ánh xạ là p_i, trong đó

$$P_i = \frac{\text{sè lÇn ¸p dông } w_i}{\text{sè lÇn ¸p dông tÊt c¶ c¸c ¸nh x¹}} \qquad \Rightarrow \sum_{i=1}^{N} P_i = 1$$

Với tập ban đầu $x_0 \in X$, chọn ngẫu nhiên chỉ số $i \in \{1, 2, ...N\}$ lặp đến khi dừng (tuỳ ý)

$x_n = w_i(x_{n-1})$ n=1, 2, 3, ... ta có $\{x_n: n=1, 2, 3, ...\} \subset X$

dãy này hội tụ tới tập hút của IFS thu được ảnh của fractal khi $n \to \infty$ sử dụng ánh xạ A phin $w_i(X)=A_i X + b_i$

$$P_i = \frac{\det(A_i)}{\sum_{j=1}^{N} \det(A_i)}$$

Giả sử {X;w_1, w_2, ..., w_N} là hypecbolic IFS, trong đó $p_i>0$ là các xác suất và $\sum_{i=1}^{N} p_i = 1$. Lấy $x_0 \in$ X và chọn liên tiếp một cách độc lập,

$x_n \in \{w_1(x_{n-1}), w_2(x_{n-1}), ..., w_N(x_{n-1}),$ n=2, 3, ... trong đó xác suất của sự kiện $x_n = w_i(x_{n-1})$ là p_i. Như thế chúng ta thu được dãy $\{x_n;$ n=0, 1, ...$\} \subset$ X. Dãy này hội tụ đến điểm hút của IFS.

CHƯƠNG II: MỘT SỐ KỸ THUẬT CÀI ĐẶT HÌNH HỌC PHÂN HÌNH.

II.1 HỌ ĐƯỜNG VONKOCK:

Trong phần này chúng ta sẽ cùng nhau thảo luận các fractal được phát sinh bằng cách sử dụng đệ qui initiator / generator với kết quả là các hình tự đồng dạng hoàn toàn. Các hình này có số chiều tự đồng dạng, số chiều fractal và số chiều Hausdorff-Besicovitch bằng nhau.

Số chiều được tính theo công thức sau:

$$D = \frac{\log(N)}{\log\left(\frac{1}{R}\right)}$$

Trong đó:

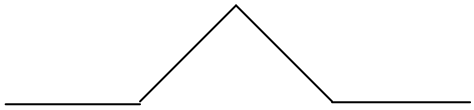
N: Là số đoạn thẳng.

R: Là số chiều dài của mỗi đoạn.

Chúng ta bắt đầu bằng một initiator, nó có thể là một đoạn thẳng hay một đa giác. Mỗi cạnh của initiator được thay thế bởi một generator, mà là tập liên thông của các đoạn thẳng tạo nên bằng cách đi từ điểm bắt đầu đến điểm cuối của đường thay thế (Thông thường các điểm của generator là một lưới vuông hay một lưới tạo bởi các tam giác đều). Sau đó mỗi đoạn thẳng của hình mới được thay thế bởi phiên bản nhỏ hơn của generator. Quá trình này tiếp tục không xác định được. Sau đây là một số đường Von Kock quan trọng:

□ ĐƯỜNG HOA TUYẾT VON KOCK-NOWFLAKE:

Đường hoa tuyết được xây dựng bởi nhà toán học Helge Von Kock vào năm 1904. Ở đây chúng ta bắt đầu với initiator là một đoạn thẳng. Còn generator được phát sinh như sau:



Generator của đường von kock

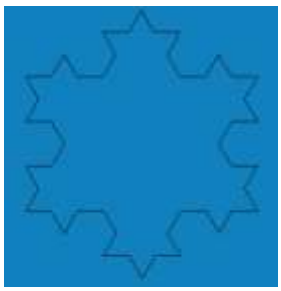
Chúng ta chia đoạn thẳng thành ba phần bằng nhau. Sau đó thay thế một phần ba đoạn giữa bằng tam giác đều và bỏ đi cạnh đáy của nó. Sau đó chúng ta lặp lại quá trình này cho mỗi đoạn thẳng mới. Nghĩa là chia đoạn thẳng mới thành ba phần bằng nhau và lặp lai các bước như trên.

Ta thấy quá trình xây dựng là tự đồng dạng, nghĩa là mỗi phần trong 4 phần ở bước thứ k là phiên bản nhỏ hơn 3 lần của toàn bộ đường cong ở bước thứ (k–1).

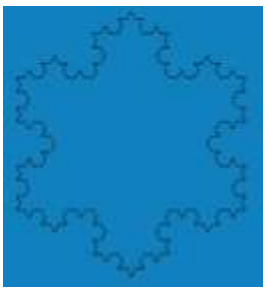
Như vậy mỗi đoạn thẳng của generator có chiều dài R = 1/3 (giả sử chiều dài đoạn thẳng ban đầu là 1) và số đoạn thẳng của generator N = 4. Do vậy số chiều fractal của đường hoa tuyết là:

$$D = \frac{\log(N)}{\log\left(\frac{1}{R}\right)} = \frac{\log 4}{\log 3} \approx 1{,}2618$$

- **Một số hình ảnh của đường**

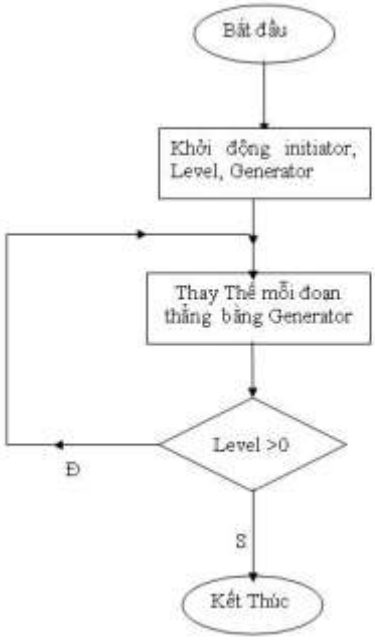


(Bậc 2)



(Bậc 3)

- **Lưu đồ thuật toán**



Mỗi lúc chúng ta thay thế đoạn thẳng bởi generator, chúng ta dùng 2 mảng XPoints, YPoints để tạo mảng các vị trí toạ độ và sau đó vẽ đoạn thẳng từ cặp tọa độ thứ nhất đến thứ hai, từ thứ hai đến thứ ba, v.v… cho đến khi chúng ta cần vẽ hết số đoạn cần vẽ NumLines (trong trường hợp đường hoa tuyết thì NumLines = 4). Để phát sinh ra các cặp tọa độ chúng ta sử dụng các lệnh đồ họa con rùa như đã mô tả ở trên.

Đầu tiên, hàm –Generator giảm Level đi một đơn vị. Sau đó chúng xác định các toạ độ của các điểm cần vẽ của generator bằng cách trước tiên tính chiều dài của mỗi đoạn thẳng của generator cần thay thế (Line-Len chính là 1/R), sau đó lưu trữ hai đầu mút của đoạn thẳng cần thay thế, rồi tính góc con rùa, sau đó di chuyển con rùa tới toạ độ đầu của đoạn thẳng này, và cuối cùng quay đi một góc thích hợp (có lúc góc quay là 0^0).

Sau đó chúng ta lặp lại quá trình sau để xác định các toạ độ của các đoạn thẳng của generator: di chuyển con rùa đi một bước, lưu trữ vị trí mới của con rùa và quay đi một góc thích hợp. Ở đây góc quay được lưu trữ trong mảng Angle. Đối với đường hoa tuyết giá trị của mảng Angle là : {0, 60, -120, 0}.

Kế tiếp hàm –Generator kiểm tra xem mức Level có lớn hơn 0 chưa:

Nếu có hàm bắt đầu lặp, xác định các toạ độ các đầu mút của đoạn thẳng mới trong các mảng toạ độ vừa mới tạo thành và sau đó gọi đệ quy hàm –Generator để thay thế mỗi đoạn bằng một generator.

Nếu Level bằng 0, hàm sẽ vẽ các đoạn thẳng được lưu trong các mảng toạ độ.

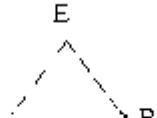
- **Code**

```
void Generator(CDC *pDC,double X1, double Y1, double X2,  double
Y2, int Level,int NumLines,double LineLen,double Angles[])
```

```
        double *XPoints,*YPoints;
        int I;
        double Turtle_Theta,Turtle_X, Turtle_Y, Turtle_R;
        XPoints = new double[NumLines +1];
        YPoints = new double[NumLines +1];
        --Level;
       Turtle_R=sqrt((X2-X1)* (X2-X1)+ (Y2-Y1)* (Y2-Y1))*LineLen;
        XPoints[0]=X1;
        YPoints[0]=Y1;
        XPoints[NumLines]=X2;
        YPoints[NumLines]=Y2;
        Turtle_Theta=Point(X1,Y1,X2,Y2);
        Turtle_X=X1;
        Turtle_Y=Y1;
        Turn(Angles[0],Turtle_Theta);
        for (I=1; I<NumLines; ++I)

                Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);
                XPoints[ I ]=Turtle_X;
                YPoints[ I ]=Turtle_Y;
                 Turn(Angles[ I ],Turtle_Theta);

            if (Level)
            for (I=0; I<NumLines; I++)

                    X1=XPoints[ I ];
                    Y1=YPoints[ I ];
                    X2=XPoints[ I +1];
                          Y2=YPoints[ I +1];
                    Generator(pDC,X1,Y1,X2,Y2,Level,
                              NumLines,LineLen,Angles);
                }
          else
            for (I= 0; I<NumLines; I++ )
            pDC->MoveTo((int)XPoints[ I ], (int) YPoints [ I ]);
            pDC->LineTo((int)XPoints[ I+1 ], (int) YPoints [ I+1 ]);
            delete[]XPoints;
            delete[]YPoints;
}
```

□ **ĐƯỜNG VON KOCK-GOSPER:**

Một dạng khác của đường Von Kock được phát hiện bởi W.Gosper. Trong đường mới này, initiator là một lục giác đều và generator chứa ba đoạn



nằm trên một lưới của các tam giác đều. Hình sau cho chúng ta thấy generator bố trí trên lưới:

Ta thấy đường này có chút khác biệt so với đường hoa tuyết ở chỗ đoạn thẳng được thay thế không nằm trên bất kỳ các đường nào của lưới.

Để tính số chiều fractal của đường Gosper trước hết ta tính chiều dài mỗi đoạn của generator. Giả sử chiều dài từ đầu mút của generator đến đầu mút khác là 1.

Đặt:

$$AC = R \Rightarrow AE = 3AC = 3R$$

$$AB^2 = AE^2 + EB^2 - 2AE.EB.\cos(60^0)$$

Ta có:

Mà AB = 1, AE = 3R, EB = AC = R

$$\Rightarrow 1 = 9R^2 + R^2 - 2*3R*R/2 = 7R^2$$

$$\Rightarrow R = \frac{1}{\sqrt{7}}$$

$$EB^2 = AE^2 + AB^2 - 2AEAB\cos\alpha$$

$$\Rightarrow \cos\alpha = \frac{AB^2 + AE^2 - EB^2}{2AEAB} = \frac{1+9R^2-R^2}{2*3R*1} = \frac{1+8R^2}{6R} = \frac{1+8\frac{1}{7}}{6\frac{1}{\sqrt{7}}} = \frac{5\sqrt{7}}{14}$$

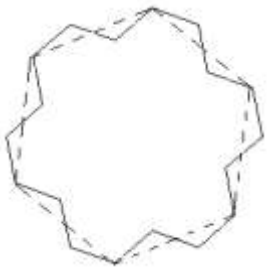
$$\approx 0\cdot 94491$$

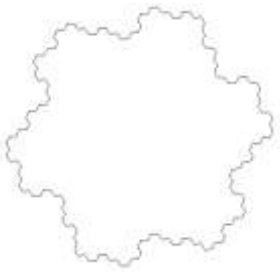
$$\Rightarrow \alpha \approx 19^0 1'$$

Vì N = 3 nên số chiều fractal của đường Gosper là:

$$D = \frac{\log 3}{\log \sqrt{7}} \approx 1.1291$$

- **Một số hình ảnh của đường**





(Mức 1) (Mức 2)

- **Code**

Đoạn mã đối với đường Gosper giống như đoạn mã của đường hoa tuyết, trong đó:

NumLines = 3

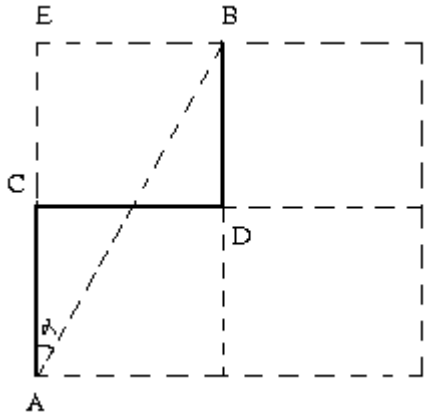
Mảng Angle có giá trị sau: {19.1, -60.0 }

Ngoài ra, đường Gosper có các mức khác nhau thì tương ứng với các hình dạng khác nhau.

□ ĐƯỜNG VON KOCK BẬC HAI 3-ĐOẠN:

Một vài đường cong kế tiếp được gọi là bậc hai (quadric) vì initiator là một hình vuông (Tuy nhiên điều này không có gì bí mật về initiator là hình vuông, nó có thể là một đa giác). Hơn nữa chúng ta sẽ tạo ra các generator trên lưới các hình vuông. Đối với đường cong đầu tiên này, một generator của 3-đoạn sẽ được sử dụng.

Hình sau sẽ cho chúng ta một generator:



Để tính số chiều fractal của đường này trước hết ta tính số chiều của mỗi đoạn của generator. Giả sử chiều dài từ đầu mút của generator đến đầu mút khác là 1:

Ta có:

Đặt AC = R

$AB^2 = AE^2 + EB^2$

Mà AB = 1, AE = 2AC = 2R, EB = R => $1 = 4R^2 + R^2$

$$=> R = \frac{1}{\sqrt{5}}$$

$EB^2 = EA^2 + AB^2 - 2EA.AB.\cos\alpha$

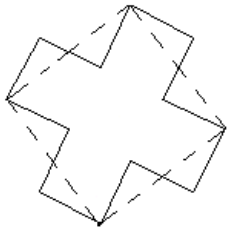
$$=> \cos\alpha = \frac{EA^2 + AB^2 - EA^2}{2EAAB} = \frac{4R^2 + 1 - R^2}{2.2R.1} = \frac{1 + 3R^2}{4R} = \frac{1 + 3\frac{1}{5}}{4\frac{1}{\sqrt{5}}} = \frac{2\sqrt{5}}{5} \approx 0.894427$$

$$\Rightarrow \alpha \approx 25^0 56'$$

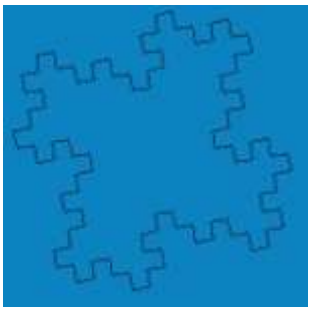
Vì N = 3 nên số chiều fractal là:

$$D = \frac{\log 3}{\log \sqrt{5}} \approx 1.3652$$

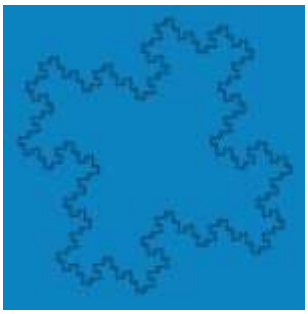
- **Một số hình ảnh của đường**



(Mức 1)



(Mức 3)



(Mức 5)

- **Code**

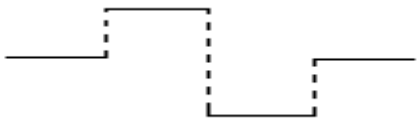
Đoạn mã đối với đường 3-đoạn giống như đoạn mã đường hoa tuyết.

NumLines = 3

Mảng Angle có giá trị sau: {26.56, -90.0 }

□ ĐƯỜNG VON KOCK BẬC HAI 8-ĐOẠN:

Một vài đường cong kế tiếp sẽ giúp sử dụng một lưới hình vuông và quay các góc đi 90^0. Chúng đều hơn một chút so với đường cong trước bởi vì đoạn thẳng được thay thế sẽ rơi vào đường nằm ngang ở giữa lưới. Hình sau cho chúng ta thấy generator của nó:



Giả sử chiều dài từ đầu nút của generator đến đầu mút khác là 1, thì chiều dài mỗi đoạn thẳng của generator R = 1/4.

Bây giờ chúng ta có thể vẽ các generator khác nhau, giới hạn duy nhất là đường cong không tự đè lên nhau và không tự giao nhau. Nếu chúng ta muốn đường cong có số chiều lớn nhất có thể có, thì chúng ta cần tìm generator với N lớn nhất. Mandelbrot đã định giá trị N lớn nhất có thể có là:

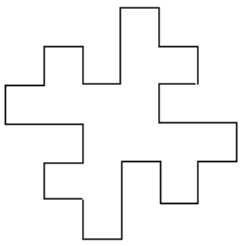
$$N_{\max} = \frac{1}{2R^2}$$ Với 1/R là số chẵn

$$N_{\max} = \frac{1+R^2}{2R^2}$$ Với 1/R là số lẻ.

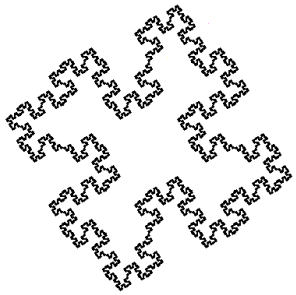
Do vậy R = 1/4 nên N_{max} = 8. Số chiều fractal là:

$$D = \frac{\log 8}{\log 4} = 1.5$$

- **Một số hình ảnh của đường**



(Mức 1)



(Mức 5)

- **Code**

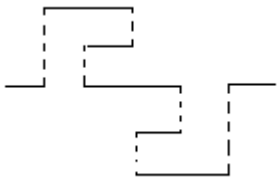
Đoạn mã đối với đường cong 8-đoạn giống như đoạn mã của đường hoa tuyết, trong đó:

NumLine = 8

Mảng Angle có giá trị sau: {0, 90, -90, -90, 0, 90, 90, 0 }

□ ĐƯỜNG VON KOCK BẬC HAI 18-ĐOẠN:

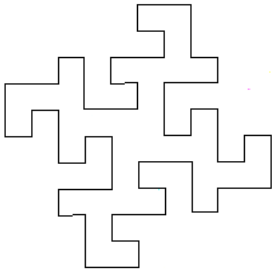
Hình sau là generator của đường Von Kock bậc hai 18-đoạn:



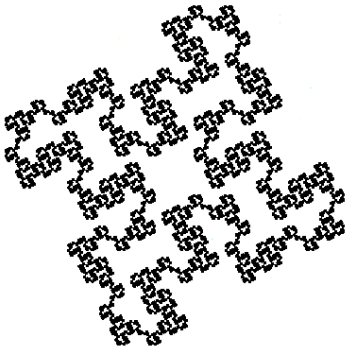
Giả sử chiều dài từ đầu mút của generator đến đầu mút khác là 1, thì chiều dài mỗi đoạn thẳng của generator là R = 1/6. Khi đó N_{max} = 18. Do đó số chiều fractal là:

$$D = \frac{\log 18}{\log 6} \approx 1.6131$$

- **Một số hình ảnh của đường**



(Mức 1)



(Mức 5)

- **Code**

Đoạn mã đối với đường 18-đoạn giống như đoạn mã của đường hoa tuyết, trong đó:

NumLine = 18

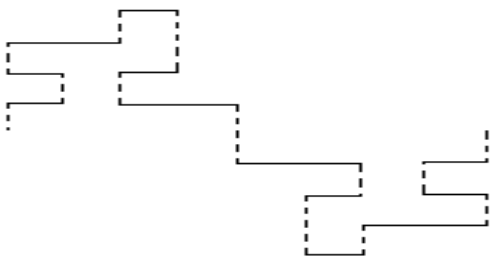
Mảng Angle có giá trị sau:

{0, 90, 0, -90, 0, -90, -90, 90, 90, 0, -90, -90, 90, 90, 0, 90, 0, 0 }

.

□ ĐƯỜNG VON KOCK BẬC HAI 32-ĐOẠN:

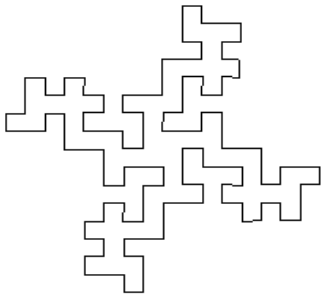
Hình sau là generator của đường Von Kock bậc hai 32-đoạn:



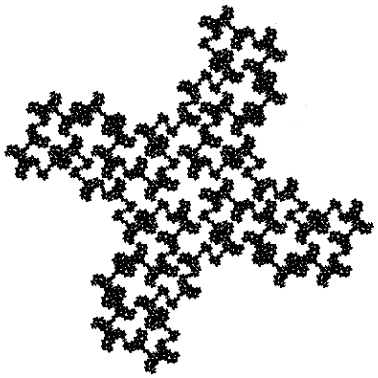
Giả sử chiều dài từ đầu mút của generator đến đầu mút khác là 1, thì chiều dài mỗi đoạn thẳng của generator là R = 1/8. Khi đó N_{max} = 32. Do đó số chiều fractal là:

$$D = \frac{\log 32}{\log 8} \approx 1.6667$$

- **Một số hình ảnh của đường**



(Mức 1)



(Mức 4)

- **Code**

Đoạn mã đối với đường 32-đoạn giống như đoạn mã của đường hoa tuyết, trong đó:

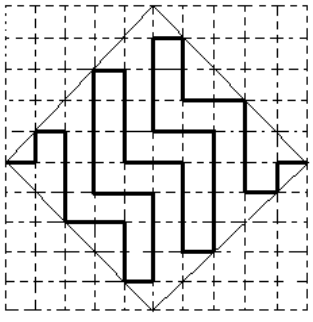
NumLine = 32

Mảng Angle có giá trị sau:

$$\begin{cases} 90,-90,90,90,-90,-90,0,90,-90,-90,0,-90,90,90,0, \\ -90,0,90,0,-90,-90,90,0,90,90,-90,0,90,90,-90,-90,0 \end{cases}$$

□ ĐƯỜNG VON KOCK BẬC HAI 50-ĐOẠN:

Hình sau là generator của đường Von Kock bậc hai 50-đoạn:



Giả sử chiều dài từ đầu mút của generator đến đầu mút khác là 1, thì chiều dài mỗi đoạn thẳng của generator là R = 1/10. Khi đó N_{max} = 50. Do đó số chiều fractal là:

$$D = \frac{\log 50}{\log 10} \approx 1.6990$$

Chúng ta thấy generator chứa nhiều đoạn thẳng hơn, do đó nó trở nên kém rõ ràng hơn trong cách thức chứa đường. Quá trình được sắp xếp và sửa sai.

Nếu chúng ta sử dụng generator để thay thế các đoạn thẳng cắt nhau theo một góc 90^0, thì chúng ta không thể có bất cứ phần nào của generator vượt ra ngoài biên của ô vuông được tạo ra bởi các đường chấm chấm (Như ở hình vẽ trên). Điều này đủ để tránh tự đè lên nhau, nhưng không ngăn cản việc tự giao nhau. Để đảm bảo ngăn chặn tự giao nhau, chúng ta nối một cách hình thức mỗi cặp cạnh song song của ô vuông. Nếu generator tiếp xúc với cạnh của ô vuông ở cùng một điểm về cùng một bên của một cặp, thì sự tự giao nhau sẽ xảy ra. Cuối cùng, cách dễ dàng để tạo ra generator là chia nó ra làm hai phần mà đối xứng với nhau, mỗi phần bắt đầu ở mút của đoạn được thay thế và kết thúc ở điểm giữa của điểm này. Do đó sự ràng buộc ở đây là:

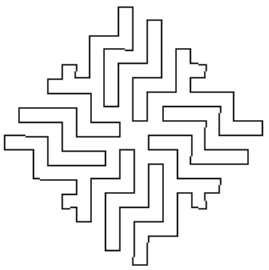
◊ Tạo một nửa generaor từ một đầu mút của đoạn được thay thế và kết thúc ở điểm giữa của đoạn này, chứa $N_{max}/2$ đoạn.

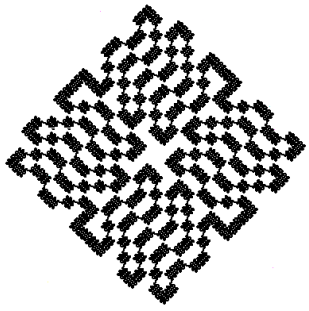
◊ Không đi ra ngoài ô vuông.

◊ Nếu generator giao với một điểm nằm trên một cặp cạnh song song với nhau của ô vuông, thì nó không thể giao nhau ở một điểm tương ứng của cặp cạnh khác.

Khi nửa generator đã tạo, chúng ta có thể lật ngược lại đồ thị và vẽ generator giống như trên để hoàn tất quá trình.

- **Một số hình ảnh của đường**





(Mức 1) (Mức 3)

- **Code**

Đoạn mã đối với đường 50-đoạn giống như đoạn mã của đường hoa tuyết, trong đó:

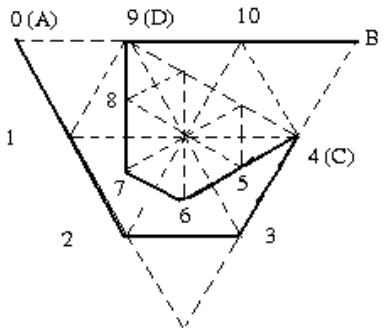
NumLine = 50

Mảng Angle có giá trị sau:

{ 0, 90, -90, -90, 0, 0, 90, 0, 0, 0, 90, 90, 0, 0, 90,
0, -90, 0, 0, 0, -90, -90, 0, 0, 90, 0, -90, 0, 0,
90, 90, 90, 0, 0, 0, 90, 0, -90, 0, 0, -90, -90, 0,
90, 0, -90, 0, 0, 90, 90, 0 }

□ GENERATOR PHỨC TẠP:

Chúng ta hãy quan sát geneator phức tạp dưới đây:



Generator này được khám phá bởi Mandelbrot. Cơ sở của nó là một lưới các tam giác đều. Nếu generator chứa các đoạn nối các điểm 0, 1, 2, 3, 4 và 11 thì nó sẽ trở nên đơn giản hơn. Tuy nhiên mô hình nhỏ hơn của generator đơn giản này được chèn vào giữa điểm 4 và 9, sau đó hai đoạn thẳng bằng nhau được thêm vào để hoàn tất generator.

Do có hai độ dài khác nhau được sử dụng, chúng ta sử dụng biểu thức sau để xác định số chiều fractal:

Ta có:

$$\sum R.M^{D} = 1$$

Trong đó:

M: Là đoạn thẳng.

R: Là chiều dài của mỗi đoạn thẳng.

D: Là số chiều của mỗi fractal.

Giả sử chiều dài từ đầu mút của generator đến đầu mút khác là 1, thì chiều dài của các đoạn đều bằng nhau là R = 1/3. Đối với các đoạn nhỏ hơn thì chiều dài là:

$$R = \frac{\sqrt{3}}{9}$$

Thật vậy:

Ta có:

$CD^2 = CB^2 + DB^2 - 2.CB.DB.\cos 60^0$

Mà:

CB = 1/3

DB = 2/3

$$CD^2 = \frac{1}{9} + \frac{4}{9} - 2 \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{1}{2}$$

$$\Rightarrow CD = \frac{\sqrt{3}}{3}$$

Do đó, chiều dài mỗi đoạn của generator nhỏ là:

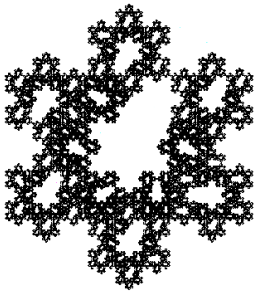
$$R = CD/3 = \frac{\sqrt{3}}{9}$$

Vậy chúng ta có:

$$6\left(\frac{1}{3}\right)^D + 5\left(\frac{\sqrt{3}}{9}\right)^D = 1$$

$$\Rightarrow D \approx 1.238361$$

- **Một số hình ảnh của đường**



(Mức 1) (Mức 3) (Mức 5)

- **Code**

Việc tạo ra đường này cũng như phần trên, nhưng hàm –Generator thì khác. Hàm này phải đảm bảo rằng đường không tự đè lên nhau và tự giao nhau. Có 4 sự thay đổi của generator ở các vị trí:

Bên phải đoạn thẳng gốc.

Bên trái đoạn thẳng gốc.

Bên phải đoạn thẳng gốc (nhưng với generator đảo ngược).

Bên trái đoạn thẳng gốc (nhưng với generator đảo ngược).

Đoạn mã của hàm –Generator này là:

//Phát sinh họ đường Von Kock Generator phức tạp:

```
void ComplexVonKockGenerator( CDC *pDC,double X1,double
        Y1[],double X2,double Y2,int Level,int Type,int
        Sign,int NumLines,double LineLen,double
        Angles[])

-
    double * XPoints ,*YPoints;
    int I;
    double Thurtle_Theta,Thurtle_X,Thurtle_Y,Thurtle_R;
    int Split=5;
    double AngleSplit=60;
    XPoints = new double[NumLines + 1];
    YPoints = new double[NumLines + 1];
    Switch(Type)
      -
        case1:
            Sign*= -1;
            break;
        case2:
            Sign*= -1;
        Case3:
         -
             Double Temp;
                        Temp = X1;
                        X1=X2;

            X2=Temp;
                        Temp = Y1;
                        Y1 = Y2;
            Y2=Temp;


            break ;


    --Level;
  Turtle_R=sqrt((X2-X1)*  (X2-X1)+  (Y2-Y1)*  (Y2-
Y1))*LineLen;
                    XPoints[0]=X1;
    YPoints[0]=Y1;
    XPoints[NumLines]=X2;
    YPoints[NumLines]=Y2;
```

```
        Turtle_Theta=Point(X1,Y1,X2,Y2);
        TurtleX=X1;
        TurtleY=Y1;
for (I=1 ; I<Split ;++I)
        -
                Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);
                        XPoints[ I ]=Turtle_X;
                YPoints[ I ]=Turtle_Y;
                Turn(Angles[ I ]*Sign,Turtle_Theta);

        for (I=NumLines -1; I>=NumLines -2; --I)
        -
                Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);


                        XPoints[ I ]=Turtle_X;
                YPoints[ I ]=Turtle_Y;
                Turn(Angles[ I ]*Sign,Turtle_Theta);


       Turtle_R=sqrt((XPoints[NumLines -2]- XPoints[Split -1])*
       (XPoints[NumLines -2]- XPoints[Split -1]) +
        (YPoints[NumLines -2]- YPoints[Split -1])*
        (YPoints[NumLines -2]- YPoints[Split -1]))*LineLen;
            Turtle_Theta= Point(XPoints[Split-1], YPoints[Split-1],
      XPoints[NumLines -2], YPoints[NumLines -2]);
                    Turn(-AngleSplit*Sign,Turtle_Theta);
        Turtle_X=XPoints [Split-1];
        Turtle_Y=YPoints [Split-1];
        for (I=Split ; I<NumLines -2 ;++I)
        -
                Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);
                XPoints[ I ]=Turtle_X;
         YPoints[ I ]=Turtle_Y;
                Turn(Angles[ I ]*Sign,Turtle_Theta);


        if(Level)
        -
                for (I=0 ; I<NumLines ;++I)
                        switch(I)
```

```
case2:
                    case8:
                    case10:
                        Type = 0;
                        break ;
                    case0:
                        Type = 1;
                        break ;
                    case5:
                    if (Level= = 1)
                        Type = 1;
                                    else
                                        Type = 3;
                        break ;
                    case1:
                    case3:
                    case4:
                        Type =2;
                        break;
                    case6:
                    case7:
                    case9:
                        Type = 3;
                        break;


            X1 = XPoints[ I ];
            Y1 = YPoints[ I ];
            X2 = XPoints[ I+1 ];
            Y2 = YPoints[ I+1 ];
ComplexVonKockGenerator(pDC,X1,Y1,X2,Y2,Level,Type,Sign,
                        NumLines,LineLen,Angles);


        else
                    for (I= 0 ; I<NumLines ; I++ )

         pDC->MoveTo((int)XPoints[ I ],(int) YPoints [ I ]);
        pDC->LineTo((int)XPoints[ I+1 ],(int) YPoints [ I+1 ]);
            delete[]XPoints;
```

```
delete[]YPoints;
```

Hàm này có thêm hai tham số Sign và Type. Sygn dùng để nhân với mỗi góc khi quay. Nếu Type = 0 không có gì thay đổi, tham số Sign vẫn duy trì giá trị củ và generator được sinh ra cùng bên như generator trước. Khi type = 1, Sign được nhân với -1 thì tất cả các góc quay theo chiều đảo ngược sao cho generator xuất hiện ở bên đối diện với đoạn thẳng từ generator trước. Khi Type = 2, chúng ta tạo đoạn thẳng mà toạ độ đầu là điểm cuối của đoạn thẳng khác và ngược lại sao cho generator được vẽ theo chiều ngược lại. Chúng ta cũng cần đảo tất cả các dấu của generator đảo ngược này để generator xuất hiện cùng bên với generator trước. Cuối cùng, khi Type = 3, chúng ta đảo ngược các toạ độ sao cho generator vừa đảo ngược vừa di chuyển sang phía đối diện. Hàm này làm việc như sau:

Xác định Type thuộc loại nào?

Xác định các toạ độ của generator lớn và nhỏ.

Nếu Level = 0 thì hàm sẽ vẽ các đoạn thẳng.

Nếu Level khác 0 thì hàm xác định loại cho tham số Type đối với mỗi đoạn thẳng, sau đó gọi đệ quy.

Mảng Angle là: {60, 0, -60, -60, -60, 0, 60, 60, 0, 0, -60}

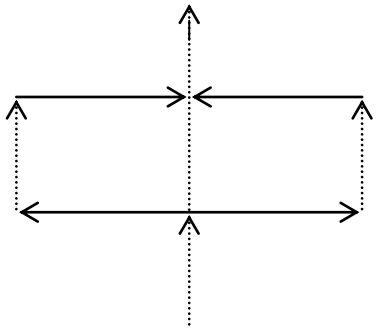
NumLines = 11

II.2 HỌ ĐƯỜNG PEANO:

Trong phần này, chúng ta xét các đường có số chiều fractal bằng 2. Chúng được gọi là các đường Peano vì đường đầu tiên trong họ đường này được khám phá bởi Guiseppe Peano vào năm 1900. Do đó chiều fractal là 2 nên các đường này phải lấp đầy hoàn toàn mặt phẳng. Điều này dẫn đến sự tự giao nhau của chúng tại nhiều điểm trong mặt phẳng.

□ ĐƯỜNG PEANO NGUYÊN THUỶ:

Hình sau cho chúng ta thấy generator của đường Peano nguyên thuỷ:



Ở đây initiator rất đơn giản. Nó chỉ là một đoạn thẳng. Thật không may, tất cả đều tự cắt, nên hầu như không thể xác định cách thức mà theo đó đường

Peano được vẽ, ngay cả các mũi tên được thêm vào trong hình vẽ. Nhìn vào hình vẽ này chúng ta thấy generator được hình thành như sau:

Đầu tiên chúng ta dựng đoạn thẳng đứng về phía trên, rồi dựng đoạn thẳng ngang về bên trái, rồi dựng đoạn thẳng đứng về phía trên, rồi dựng dựng đoạn thẳng ngang về bên phải, rồi dựng đoạn thẳng đứng về phía dưới, rồi dựng đoạn thẳng ngang về bên phải, rồi dựng đoạn thẳng đứng về phía trên, rồi dựng đoạn thẳng ngang về phía trái, và cuối cùng dựng đoạn thẳng đứng về phía trên.

Như vậy generator chứa 9 đoạn thẳng (nghĩa là N = 9), chiều dài mỗi đoạn của generator là R = 1/3 (Giả sử chiều dài đoạn thẳng ban đầu là 1). Do đó số chiều fractal là:

$$D = \frac{\log 9}{\log 3} \Rightarrow D = 2$$

- **Một số hình ảnh của đường**

(Mức 1) (Mức 3)

- **Code**

Đoạn mã đối với đường Peano giống đoạn mã của đường hoa tuyết, trong đó:

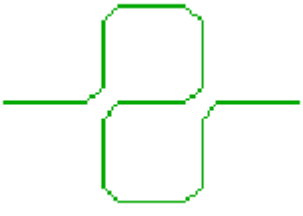
NumLines = 9

Mảng Angle có giá trị sau:

0;90,–90,–90,–90,90,90,90,0

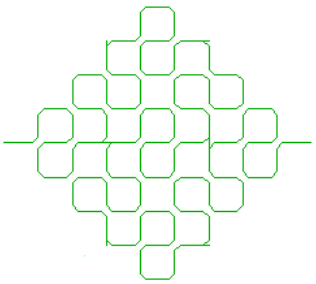
□ ĐƯỜNG PEANO CẢI TIẾN:

Nếu không có sự tự giao của generator đối với đường Peano thì việc đi theo vết của nó và quan sát cách thức vẽ sẽ dễ dàng hơn. Vì thế, đường Peano cải tiến được phát triển theo kiểu làm tròn các góc để tránh sự tự giao. Kết quả chúng ta được generator như hình sau:



Tuy nhiên, generator cập nhật này chỉ có thể sử dụng ở mức thấp nhất trước khi thực vẽ đường cong. Nếu sử dụng nó ở mức cao hơn, bằng kỹ thuật đệ quy chúng ta cố gắng thay thế generator đối với mỗi đường chéo được làm tròn ở một góc, cũng như đối với các đoạn thẳng đều. Do đó generator cho đường Peano nguyên thuỷ được sử dụng ở mức cao. Vì generator sử dụng lần đệ quy cuối cùng có độ dài ngắn hơn so với đường Peano nguyên thuỷ, ta có số chiều fractal D nhỏ hơn 2. Khi số lần đệ quy tăng lên, số chiều fractal sẽ thay đổi và tiến về 2.

- **Một số hình ảnh của đường**



(Mức 2)

- **Code**

// Edit Code phát sinh của đường cong Peano cải tiến:

```
void ModifiedPeanoGenerator(CDC *pDC, double X1, double Y1,
                double X2, double Y2, int Level, int NumLines,
                double LineLen, double Angles[],
                double &XTemp, double &YTemp)

-
        double *XPoints, *YPoints,SplitLineLen=1.0/18.0;
         int I,Split = 9;
               double Turtle_Theta,Turtle_X, Turtle_Y, Turtle_R;
         XPoints = new double[NumLines + 1];
         YPoints = new double[NumLines + 1];
--Level;
XPoints[0]= X1;
         YPoints[0]= Y1;
         Turtle_Theta = Point(X1,Y1,X2,Y2);
```

```
Turtle_X=X1;
        Turtle_Y=Y1;


if (Level)
  -
          Turtle_R=sqrt((X2-X1)* (X2-X1)+
                        (Y2-Y1)* (Y2-Y1))*LineLen;
                  XPoints[Split]=X2;
          YPoints[Split]=Y2;
            for (I=1; I<Split; ++I)
              -
              Step(Turtle_X,          Turtle_Y,          Turtle_R,
              Turtle_Theta);
                        XPoints[ I ]=Turtle_X;
                  YPoints[ I ]=Turtle_Y;
                  if (I!=Split)
                        Turn(Angles[ I ],Turtle_Theta);


              for (I=0 ; I<Split ;++I)

              -
                                    X1 = XPoints [ I ];

                                    Y1 = YPoints [ I ];
                              X2 = XPoints [ I+1 ];
                                    Y1 = YPoints [ I+1 ];
                        ModifiedPeanoGenerator(pDC,    X1,    Y1,
                                    X2,  Y2,  Level,   NumLines,
                                    LineLen,  Angles,  XTemp,
                                    YTemp);



  -
else
          Turtle_R=sqrt((X2-X1)* (X2-X1)+ (Y2-Y1)*
                                    (Y2- Y1))*SplitLineLen ;
                  XPoints[0]= XTemp;
          YPoints[0]= YTemp;
          XPoints[NumLines]= X2;
          YPoints[NumLines]= Y2;

          for (I=1; I<NumLines; ++I)
            -
```

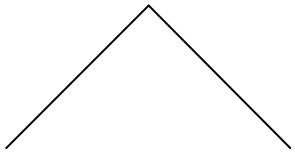
```
                    Step(Turtle_X,      Turtle_Y,      Turtle_R,
                                                Turtle_Theta);
                    XPoints[ I ]=Turtle_X;
                    YPoints[ I ]=Turtle_Y;
                    if ( I= = NumLines -1)
                            break;
                    if ( I%2)
                     -
                            Step(Turtle_X,  Turtle_Y,  Turtle_R,
                                                Turtle_Theta);
                            Step(Turtle_X,  Turtle_Y,  Turtle_R,
                                                Turtle_Theta);
                            Step(Turtle_X,  Turtle_Y,  Turtle_R,
                                                Turtle_Theta);


                    else
                     -
                            Step(Turtle_X,  Turtle_Y,  Turtle_R,
                                                Turtle_Theta);
                                    Turn(Angles[            I/2
                     -
];Turtle_Theta);


                        XTemp = XPoints [NumLines -1];
                        YTemp = YPoints [NumLines -1];
                       for (I= 0 ; I<NumLines ;++I)
                        -
                       pDC->MoveTo((int)XPoints[ I ],(int)YPoints[ I ]);
                  pDC->LineTo((int)XPoints[ I+1 ],(int)YPoints[ I+1 ]);


                       delete[]XPoints;
                          delete[]YPoints;
```

Đối với Level > 1 thì hàm này giống như hàm –Generator của đường Peano gốc. Với Level = 1, thì hàm này có hơi khác một chút. Thay vì định nghĩa bước con rùa (là biến Turtle_R) bằng 1/3 chiều dài đoạn thẳng ban đầu thì ta định nghĩa nó bằng 1/18 chiều dài đoạn thẳng ban đầu, về cơ bản generator được viết sao cho con rùa vẫn đi qua con đường giống như generator của đường Peano gốc, sử dụng các góc quay giống nhau, nhưng dùng 6 bước thay vì 1 bước như generator của Peano gốc. Tuy nhiên, các điểm được lưu trữ trong các mảng toạ độ có thay đổi. Sau khi lưu trữ toạ độ thứ nhất, ta lưu trữ vị trí sau bước 5, vị trí kế tiếp được lưu trữ ở cuối bước 1 sau khi quay đi một góc

đầu tiên. Các vị trí còn lại được lưu trữ sau bước 5 và sau bước đầu tiên của đoạn thẳng kế, ngoại trừ bước 5 của đoạn thẳng cuối cùng sẽ không được lưu trữ lại. Kết quả là khi các đoạn thẳng được vẽ, chúng sẽ tạo nên một đường xiên nối các điểm 1/6 khoảng cách trên mỗi đoạn thẳng gặp nhau ở góc. (Ở đây NumLines = 19).

□ TAM GIÁC CESARO:

Hình sau cho chúng ta xem một generator rất đơn giản (initiator là đoạn thẳng nằm ngang):



Generator chứa hai cạnh của một tam giác cân. Do đó, số đoạn thẳng là N=2 và chiều dài của mỗi đoạn là:

$$R = \frac{1}{\sqrt{2}}$$

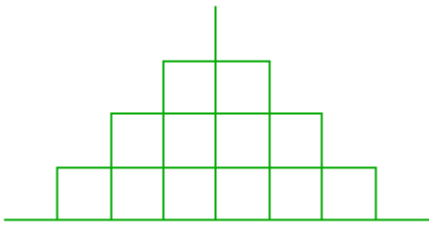
Giả sử đoạn thẳng ban đầu có chiều dài là 1. Khi đó số chiều fractal là:

$$D = \frac{\log 2}{\log \sqrt{2}} \Rightarrow D = 2$$

Phụ thuộc vào các điều kiện cụ thể, generator này sẽ được đặt bên trái hoặc bên phải của mỗi đoạn thẳng mà nó thay thế. Nhiều đường cong khác nhau hoàn toàn có thể được sinh ra từ generator này. Các đường này được khám phá bởi Ernest Cesaro vào năm 1905.

Các hình sau là các mức khác nhau của tam giác Cesaro:

Mức thứ nhất



Mức thứ năm

Ở bất kỳ mức nào trong việc xây dựng đường này, generator luôn được đặt ở bên phải của mỗi đoạn thẳng ở mức đầu tiên, bên trái của mỗi đoạn thẳng ở mức thấp hơn kế tiếp, bên phải của mỗi đoạn thẳng ở mức thấp hơn kế tiếp nữa v.v…

Như vậy đoạn mã của hàm Generator có thêm mảng Sign để lúc quay theo các góc khác nhau, chúng ta nhân tương ứng với phần tử của mảng Sign được khởi động như sau:

```
for (I = Level; I >=0; --I )
  {
     Sign [ I ] = Sign1;
     Sign1 = -1;
  }
```

Với Sign1 có thể khởi động lúc đầu là 1 hoặc -1.

// Sau đây là Edit Code của đường cong Tam Giác Cesaro.

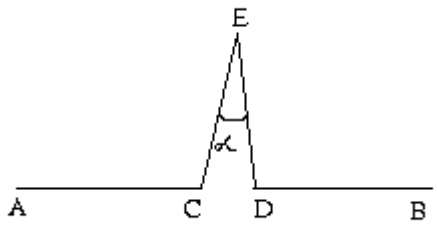
```
void CesaroTriangleGenerator(CDC *pDC,double X1, double Y1, double X2, double Y2, int Level,int NumLines, double LineLen, double Angles[],int Sign[])
-
    double *XPoints ,*YPoints;
        int I;
    double Turtle_Theta,Turtle_X, Turtle_Y, Turtle_R;
    XPoints = new double [NumLines + 1];
    YPoints = new double [NumLines + 1];
    --Level;
      Turtle_R=sqrt((X2-X1)* (X2-X1)+ (Y2-Y1)* (Y2-Y1))*LineLen;
        XPoints[0]= X1;
    YPoints[0]= Y1;
    XPoints[NumLinesv -1]= X2;
    YPoints[NumLines -1]= Y2;
    Turtle_Theta = Point(X1,Y1,X2,Y2);
    Turtle_X=X1;
```

```
                Turtle_Y=Y1;
                 for (I=NumLines ; I>=1 ;I-=2)
                 -
                          Step(Turtle_X, Turtle_Y, Turtle_R,
Turtle_Theta);
                           XPoints[ I ]=Turtle_X;
                        YPoints[ I ]=Turtle_Y;
                        Turn(Angles[ I ]*Sign[Level],Turtle_Theta);


              if (Level)
                 for ( I=0 ; I<NumLines -1 ;++I )
                 -
                        X1 = XPoints [ I ];
                        Y1 = YPoints [ I ];
                        X2 = XPoints [ I+1 ];
                        Y2 = YPoints [ I+1 ];
                        CesaroTriangleGenerator(pDC, X1, Y1, X2,
                              Y2Level, NumLines, LineLen, Angles,
                              Sign );
            else


                 -
                 for ( I= 0; I<NumLines;++I )

                     pDC->MoveTo((int)XPoints[ I ],(int)YPoints[ I ]);
                   pDC->LineTo((int)XPoints[ I+1 ],(int)YPoints[ I+1 ]);


         delete[]XPoints;
                  delete[]YPoints;
```

□ **TAM GIÁC CESARO CẢI TIẾN:**

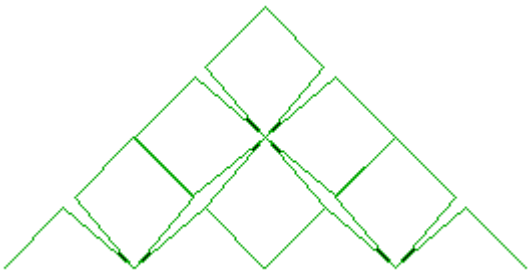
Tam giác mô tả ở trên hơi khó để lần theo dấu vết vì đường rẽ theo góc 90^0 từ tâm của đoạn thẳng gốc thật sự đi lại theo vết của nó nhưng không thể quan sát được khi vẽ. Việc cập nhật đường cesaro có thể thực hiện bằng cách thay đổi góc generator từ 90^0 sang 85^0 đối với mức thấp nhất trước khi thực hiện quá trình vẽ.

Hình sau minh hoạ một generator (initiator là đoạn thẳng nằm ngang).



Giống như đường Peano cải tiến, kết quả thu được là một đường cong mà số chiều fractal không hoàn toàn là 2, nhưng khi số lần đệ quy tiến ra vô cực thì số chiều fractal tiến về 2.

Hình sau cho chúng ta thấy mức thứ tư của tam giác Cesaro cải tiến:



Ta tính chiều dài mỗi đoạn của generator.

Giả sử chiều dài đoạn thẳng gốc là a

Ta có:

AE = a / 2

Đặt:

AC = b

CD = c

Ta có:

$$c^2 = CE^2 + DE^2 - 2CEDE\cos\alpha$$

Mà

$$CE = DE = a/2, \alpha = 10^0$$

$$\Rightarrow c^2 = \frac{a^2}{2} - 2\cdot\frac{a}{2}\cdot\frac{a}{2}\cdot\cos 10^0 = \frac{a^2}{2}(1-\cos 10^0) = a^2(\sin 5^0)^2$$

$$\Rightarrow c = a.\sin 5^0$$

Ta có:

$$AC + CD + DB = AB$$

$$\Rightarrow 2b + c = a$$

$$\Rightarrow 2b = a(1-\sin 5^0)$$

$$\Rightarrow b = \frac{a}{2}(1-\sin 5^0) = \frac{a}{2} * 0.9128442$$

Đoạn mã của hàm –Generator như sau:

```
void ModifiedCesaroGenerator(CDC *pDC,double X1, double Y1,
                             double X2, double Y2, int Level,int
                             NumLines, double LineLen, double
                             Angles[],int Sign[])
        double *XPoints , *YPoints ;
        int I;
-
        double Turtle_Theta,Turtle_X, Turtle_Y, Turtle_R,Turtle_R1;
        XPoints = new double [NumLines + 1];
        YPoints = new double [NumLines + 1];
        --Level;
        Turtle_R1=sqrt((X2-X1)*  (X2-X1)+  (Y2-Y1)*  (Y2-
Y1))*LineLen;
        Turtle_R=Turtle_R1* 0.9128442;
    XPoints [0]= X1;
    YPoints [0]= Y1;
    XPoints[NumLines -2]= X2 ;
    YPoints[NumLines -2]= Y2 ;
    Turtle_Theta = Point(X1,Y1,X2,Y2);
    Turtle_X=X1;
        Turtle_Y=Y1;
        for (I=NumLines -1; I >=1; I-=2)
        -
                Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);
                    XPoints[ I ]=Turtle_X;
                YPoints[ I ]=Turtle_Y;
                Turn(Angles[ I ]* Sign[Level],Turtle_Theta);
                if (I= =NumLines - 1)
                Turtle_R=Turtle_R1;
        Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);

        XPoints[NumLines ]=Turtle_X;
        YPoints[NumLines ]=Turtle_Y;
        Turn(Angles[NumLines ]* Sign[Level],Turtle_Theta);
        if (Level)
            for (I= 0; I<NumLines - 2; ++I)
            -
        X1 = XPoints [ I ];
                Y1 = YPoints [ I ];
                X2 = XPoints [ I+1 ];
                Y2 = YPoints [ I+1 ];
```

```
                    ModifiedCesaroGenerator(pDC, X1, Y1, X2,
                        Y2,Level,NumLines, LineLen, Angles,Sign);

            else

            -
              for (I= 0; I<NumLines -2; ++I)

              -
                    pDC->MoveTo((int)XPoints[ I ],(int)YPoints[ I ]);
                      pDC->LineTo((int)XPoints[  I+3  ],(int)YPoints[  I+3

        ]);
              for (I= 1; I<NumLines -1; ++I)
                          pDC->MoveTo((int)XPoints[  I  ],(int)YPoints[  I
]);
              -


                    pDC->LineTo((int)XPoints[ I+2 ],(int)YPoints[ I+2 ]);
             delete[]XPoints;
                  delete[]YPoints;
```

Hàm này giống với hàm trong đường Cesaro gốc nhưng lúc này mảng Angle là {0, -170, 0, 85, 0 } và NumLines = 4, đồng thời chiều dài các đoạn của generator có khác nhau. Chúng chia làm hai loại:

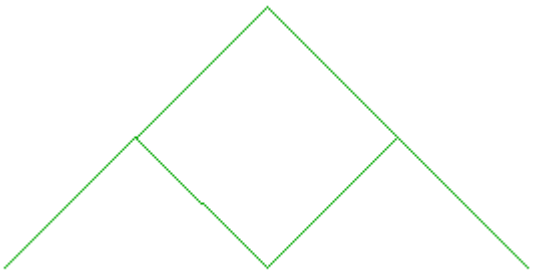
Loại chiều dài thứ nhất: Bằng nửa chiều dài của đoạn thẳng ban đầu.

Loại chiều dài thứ hai: Bằng nửa chiều dài của đoạn ban đầu nhân với 0.9128442.

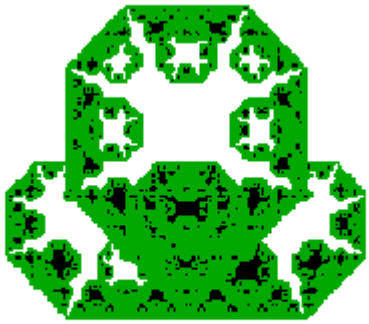
□ MỘT DẠNG KHÁC CỦA ĐƯỜNG CESARO:

Giả sử chúng ta bắt đầu với đường generator và hai mức đầu tiên như ở đường Cesaro, nhưng sử dụng sự sắp xếp khác đi khi đặt generator về phía bên trái và phải của đoạn thẳng gốc khi chúng ta ở mức cao hơn. Kết quả là nhiều đường khác nhau có thể được sinh ra từ cách sắp xếp này.

Hình sau cho chúng ta mức khác nhau của hình Cesaro này:



Mức thứ 2



Mức thứ 16

Đoạn mã của hàm –Generator như sau:

```
void OtherCesaroGenerator(CDC *pDC,double X1, double Y1, double
                          X2,  double Y2, int Level,int NumLines,
                          double LineLen, double Angles[],int Sign)

-
        double *XPoints ,*YPoints;
        int I;
  double Turtle_Theta,Turtle_X, Turtle_Y, Turtle_R;
        XPoints = new double [NumLines + 1];
        YPoints = new double [NumLines + 1];
        --Level;
 Turtle_R=sqrt((X2-X1)* (X2-X1)+ (Y2-Y1)* (Y2-Y1))*LineLen;
        XPoints[0]= X1;
  YPoints[0]= Y1;

        XPoints[NumLinesv -1]= X2;
        YPoints[NumLines -1]= Y2;
 Turtle_Theta = Point(X1,Y1,X2,Y2);
```

```
        Turtle_X=X1;
        Turtle_Y=Y1;
                    for (I=NumLines; I>=1; I-=2)
                    -
                            Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);
                                XPoints[ I ]=Turtle_X;
                            YPoints[ I ]=Turtle_Y;
                            Turn(Angles[ I ]*Sign,Turtle_Theta);


                    Sign= -1;
                if (Level)
                    for(I=0; I<NumLines -1; ++I)
                    -
                            X1 = XPoints [ I ];
                            Y1 = YPoints [ I ];
                            X2 = XPoints [ I+1 ];
                            Y2 = YPoints [ I+1 ];
                            OtherCesaroGenerator(pDC, X1, Y1, X2, Y2,
                                        Level,NumLines,LineLen,
                                        Angles,Sign );

                  else
                    for( I= 0; I<NumLines; ++I )
                      -
                                pDC->MoveTo((int)XPoints[ I ],(int)YPoints[ I
]);
                          pDC->LineTo((int)XPoints[ I+1 ],(int)YPoints[ I+1 ]);


                delete[]XPoints;
                      delete[]YPoints;

    }
```

□ **TAM GIÁC POLYA:**

Đường này được khám phá bởi George Polya. Initiator và generator thì giống như đường Cesaro, nhưng vị trí đặt chúng có thay đổi.

Hình sau cho chúng ta thấy hai mức đầu tiên của tam giác Polya:

Giống như đường Cesaro, vị trí của generator đầu tiên thay đổi từ phải sang trái và được bắt đầu ở mức đầu tiên. Đối với đường này, vị trí của

generator cũng thay đổi đường so với mỗi đoạn thẳng tương đương với các mức khác nhau:

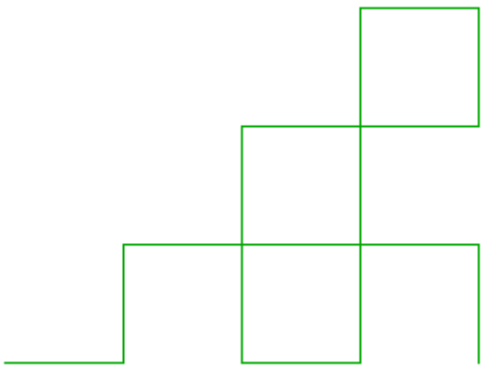
Đoạn mã của hàm Polya-Generator như sau:

```
void PolyaGenerator(CDC *pDC,double X1, double Y1, double X2,
                    double Y2, int Level,int NumLines, double
                    LineLen, double Angles[],int Sign[])

-

    double *XPoints ,*YPoints;
    int I;
  double Turtle_Theta,Turtle_X, Turtle_Y, Turtle_R;
    XPoints = new double [NumLines + 1];
    YPoints = new double [NumLines + 1];

    Turtle_R=sqrt((X2-X1)*  (X2-X1)+  (Y2-Y1)*  (Y2-
Y1))*LineLen;
        XPoints[0]= X1;
    YPoints[0]= Y1;
    XPoints[NumLinesv -1]= X2;
    YPoints[NumLines -1]= Y2 ;
  Turtle_Theta = Point(X1,Y1,X2,Y2);
  Turtle_X=X1;
    Turtle_Y=Y1;
    Turn(Angles[ 0 ]*Sign[Level],Turtle_Theta);
      for (I=1 ; I<NumLines ;++I)

      -

          Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);
            XPoints[ I ]=Turtle_X;
          YPoints[ I ]=Turtle_Y;
          Turn(Angles[ I ]*Sign[Level],Turtle_Theta);


      --Level;
    if (Level)
      for (I=0; I<NumLines; ++I)

      -

          X1 = XPoints[ I ];
          Y1 = YPoints[ I ];
          X2 = XPoints[ I+1 ];
          Y2 = YPoints[ I+1 ];
```

```
                    PolyaGenerator(pDC, X1, Y1, X2, Y2, Level,
                    NumLines, LineLen, Angles, Sign );


                    Sign[Level]* = -1;
            else
                for (I= 0; I<NumLines; ++I)

                    pDC->MoveTo((int)XPoints[ I ],(int)YPoints[ I
]);
                    pDC->LineTo((int)XPoints[ I+1 ],(int)YPoints[ I+1

    ]);
                delete[]XPoints ;
                        delete[]YPoints ;
```

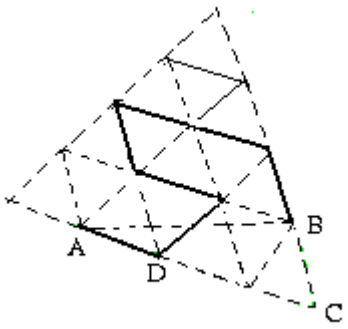
Chúng ta sử dụng cùng một kỹ thuật đã được sử dụng đối với đường Cesaro tức là cũng dùng mảng Sign sau khi chúng ta gọi đệ quy hàm – Generator. Mảng Angles có giá trị là {45, 0 } và NumLines = 2.

Tuỳ vào mức khác nhau thì tương ứng với hình vẽ khác nhau. Sau đây là hình minh hoạ của đường Polya có mức là 4:



□ ĐƯỜNG PEANO_GOSPER:

Hình sau là generator của đường Peano_Gosper và một lưới gồm các tam giác đều liên kết với nó (initiator là một đoạn thẳng nằm ngang):



Giả sử chiều dài từ đầu mút của generator đến đầu mút khác là 1, chúng ta tính chiều dài mỗi đoạn của generator như sau:

Đặt:

AD = R

Ta có:

$AB^2 = AC^2 + BC^2 - 2.AC.BC.\cos 60^0$

Mà: AC = 3AD = 3R

BC = AD = R

AB = 1

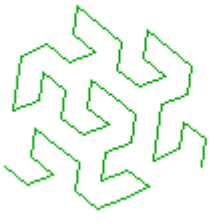
$$\Rightarrow 1 = 9 * R^2 + R^2 - 2 * 3R * R / 2 = 7R^2$$

$$\Rightarrow D = \frac{1}{\sqrt{7}}$$

Vì generator có số đoạn thẳng N = 7 nên số chiều fractal là:

$$D = \frac{\log 7}{\log \sqrt{7}} \Rightarrow D = 2$$

Đường này có tính chất là nó lấp đầy phần bên trong của đường Gosper. Hình sau cho chúng ta thấy mức thứ hai của đường này:



Đoạn mã của hàm Peano-Gosper-Generator:

```
void PeanoGosperGenerator(CDC *pDC, double X1, double Y1, double
                          X2, double Y2, int Level, int Type, int
                          NumLines, double LineLen, double Angles[
                          ])
-
        double * XPoints ,*YPoints;
        int I;
        int Sign =1;
        double Thurtle_Theta,Thurtle_X,Thurtle_Y,Thurtle_R;
        XPoints = new double[NumLines + 1];
        YPoints = new double[NumLines + 1];
                Switch(Type)
                  -
                        case 0:
                                break;
                        case 1:
                                Sign*= -1;
                                break;
                        case 2:

                Sign*= -1;
                        Case 3:
                          -
                                Double Temp;
                                Temp = X1;
                                X1=X2;
                                X2=Temp;
                                Temp = Y1 ;
                                Y1 = Y2 ;
                                Y2=Temp;
                                break ;



                --Level;
              Turn_R=sqrt((X2-X1)*      (X2-X1)+      (Y2-Y1)*      (Y2-
Y1))*LineLen;
                XPoints[0]=X1;
                YPoints[0]=Y1;
                XPoints[NumLines]=X2;
                YPoints[NumLines]=Y2;
                Turtle_Theta=Point(X1,Y1,X2,Y2);
                TurtleX=X1;
                TurtleY=Y1;
                Turn(Angles[ 0 ]*Sign,Turtle_Theta);
```

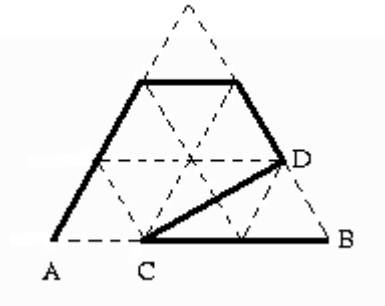
```
for (I=1; I<NumLines; ++I)
-
        Step(Turtle_X, Turtle_Y, Turtle_R,
Turtle_Theta);
            XPoints[ I ]=Turtle_X;
        YPoints[ I ]=Turtle_Y;
        Turn(Angles[ I ]*Sign,Turtle_Theta);


if(Level)
        for (I=0 ; I<NumLines ;++I)
        -
            switch(I)
            -
                                        case 0:
                    case 3:
                    case 4:
                    case 5:
                        Type = 0;
                        break;
                    case 2:
                    case 1:
                    case 6:
                        Type = 3;
                        break;


                        X1 = XPoints[ I ];
                    Y1 = YPoints[ I ];
        X2 = XPoints[ I+1 ];
        Y2 = YPoints[ I+1 ];
        PeanoGosperGenerator(pDC,X1,Y1,X2,Y2,Level,
                Type,NumLines,LineLen,Angles);


else
            for (I= 0 ; I<NumLines ; I++ )
        -
       pDC->MoveTo((int)XPoints[ I ],(int) YPoints [ I ]);
       pDC->LineTo((int)XPoints[ I+1 ],(int) YPoints [ I+1
]);
delete[]XPoints;
delete[]YPoints;
```

Hàm này cũng giống như hàm –Generator của đường Gosper, chỉ khác là nó thêm hai tham số Sign và Type như trong họ các Generator phức tạp được trình bày trong phần Complex Von Kock Generator trước. Ở đây NumLines = 7 và mảng Angles là:

$$-19.1,60,120,-60,-120,0,0$$

□ ĐƯỜNG HOA TUYẾT PEANO 7-ĐOẠN:

Hình sau là generator của đường hoa tuyết Peano 7-đoạn (initiator là một đoạn nằm ngang):



Đường này được khám phá bởi Mandelbrot. Chú ý rằng tương tự như generator sử dụng trong mục "Generator phức tạp" của họ đường Von Kock đã trình bày ở phần II.1. Điểm khác nhau duy nhất là generator này không chứa các mô hình nhỏ hơn.

Giả sử chiều dài từ đầu mút của generator đến đầu mút khác là 1, chúng ta tính chiều dài mỗi đoạn của generator.

Ta thấy, generator có 7 đoạn, trong đó có 6 đoạn có chiều dài bằng nhau là R = 1/3, còn chiều dài của đoạn còn lại là:

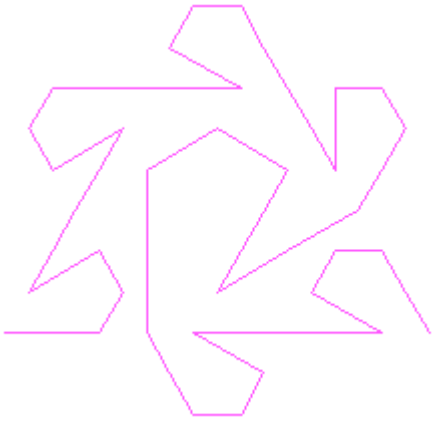
$$R = \frac{\sqrt{3}}{3}$$

(theo cách tính ở phần generator phức tạp).

Do đó:

$$6 * \left(\frac{1}{3}\right)^D + \left(\frac{\sqrt{3}}{3}\right)^D = 1 \Rightarrow D = 2$$

Hình sau cho chúng ta thấy mức thứ hai của đường hoa tuyết Peano 7-đoạn này:



Đoạn mã của đường hoa tuyết Peano 7-đoạn:

```
void Peano7-DoanGenerator(CDC *pDC,double X1, double Y1 • ,
                          double X2, double Y2, int Level, int Type,
                          int Sign, int NumLines, double LineLen,
                          double Angles[])

-
        double * XPoints ,*YPoints;
        int I;
        double Thurtle_Theta,Thurtle_X,Thurtle_Y,Thurtle_R;
        XPoints = new double[NumLines + 1];
        YPoints = new double[NumLines + 1];
        Switch(Type)
          -
            Case 0:
                break;
            case 1:
                Sign*= -1;
                break;
            case 2:
                Sign*= -1;
            Case 3:
              -
                                Double Temp;
                                Temp = X1;
                                X1=X2;

                X2=Temp;
                                Temp = Y1 ;
                                Y1 = Y2 ;
                Y2=Temp;
                break ;
```

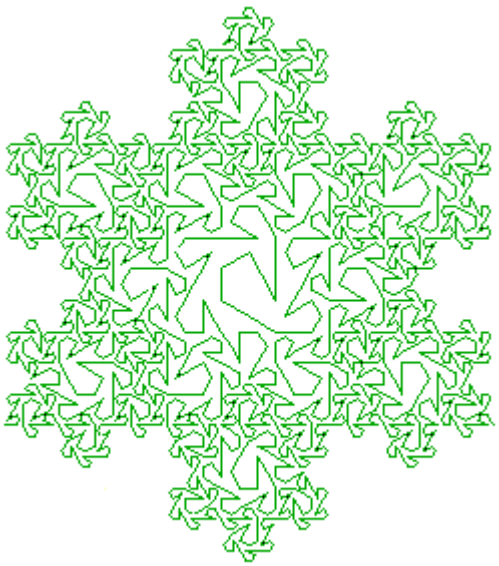
```
--Level;

Turtle_R=sqrt((X2-X1)* (X2-X1)+ (Y2-Y1)* (Y2-Y1))*LineLen;
XPoints[0]=X1;
YPoints[0]=Y1;
XPoints[NumLines]=X2;
YPoints[NumLines]=Y2;
Turtle_Theta=Point(X1,Y1,X2,Y2);
Turtle_X=X1;
Turtle_Y=Y1;
Turn(Angles[ 0 ]*Sign,Turtle_Theta);
for (I=1; I<NumLines -2; ++I)
-
Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);
XPoints[ I ]=Turtle_X;
YPoints[ I ]=Turtle_Y;
Turn(Angles[ I ]*Sign,Turtle_Theta);

for (I=NumLines -1; I>=NumLines -2; --I)
-
Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);
XPoints[ I ]=Turtle_X;
YPoints[ I ]=Turtle_Y;
Turn(Angles[ I ]*Sign,Turtle_Theta);

if(Level)
for (I=0; I<NumLines; ++I)
-
switch(I)
-
case 0:
case 5:
Type =1;
break;
case 1:
case 2:
case 3:
case 6:
```

```
                                        Type = 2;
                                        break;
                                case 4:
                                        Type = 3;
                                        break;


                                    X1 = XPoints[ I ];
                                Y1 = YPoints[ I ];
                        X2 = XPoints[ I+1 ];
                        Y2 = YPoints[ I+1 ];
                Peano7-DoanGenerator(pDC,X1,Y1,X2,Y2,Level,Type,
                                        Sign,NumLines,LineLen,Angles);


                else
                   for (I= 0 ; I<NumLines ; I++ )
                      -
                       pDC->MoveTo((int)XPoints[ I ],(int) YPoints [ I ]);
                       pDC->LineTo((int)XPoints[ I+1 ],(int) YPoints [ I+1
]);


                delete[]XPoints;
                delete[]YPoints;
```

Giống như trường hợp generator phức tạp, có 4 khả năng lựa chọn cho các vị trí của generator và phải chọn một cách cẩn thận ở mỗi mức, mỗi đoạn thẳng để đảm bảo rằng đường cong được tạo thành không tự giao nhau hay tự chồng lên nhau. Ở đây NumLines = 7 và mảng Angles là:

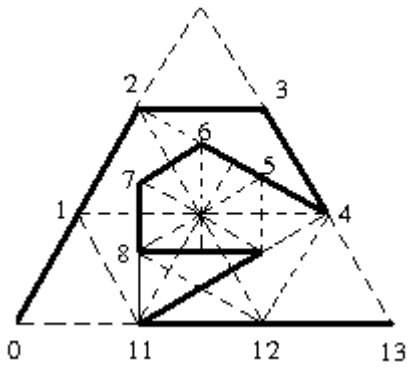
60,0,–60,–60,–60,0,–60

Tuỳ vào mức khác nhau thì tương ứng với hình vẽ khác nhau. Sau đây là hình minh hoạ của đường Peano 7-đoạn có mức là 4:



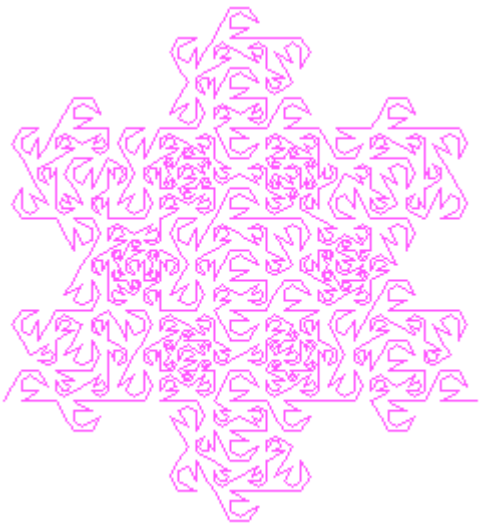
□ ĐƯỜNG HOA TUYẾT PEANO 13-ĐOẠN:

Hình sau thể hiện generator của đường hoa tuyết Peano 13-đoạn (initiator là một đoạn nằm ngang):



Đường này cũng được khám phá bởi Mandelbrot. Generator này thu được bằng cách thay thế đoạn thứ 5 của generator đường hoa tuyết 7-đoạn với mô hình nhỏ hơn của toàn bộ generator đường hoa tuyết 7 đoạn. Để tính số chiều fractal của đường này, chúng ta sử dụng công thức ở mục về đường hoa tuyết 7-đoạn. Do đó số chiều fractal của đường này vẫn là 2.

Hình sau cho chúng thấy mức thứ ba của đường hoa tuyết Peano 13-đoạn này:



Đoạn mã của đường hoa tuyết Peano 13-đoạn như sau:

```
void Peano13-DoanGenerator(CDC *pDC, double X1, double Y1,
                           double X2, double Y2, int Level, int Type,
                           int Sign, int NumLines, double LineLen,
                           double Angles[])
-
        double * XPoints ,*YPoints;
        int I;
        int Split =5;
        double AngleSplit= 60;
        double Thurtle_Theta,Thurtle_X,Thurtle_Y,Thurtle_R;
        XPoints = new double[NumLines + 1];
        YPoints = new double[NumLines + 1];
        Switch(Type)
          -
              Case 0:
                    break;
              case 1:
                    Sign*= -1;
                    break;
              case 2:
                    Sign*= -1;
              Case 3:
                -
                                  Double Temp;
                                  Temp = X1;
                                  X1=X2;

                    X2=Temp;
                                  Temp = Y1 ;
                                  Y1 = Y2 ;
                    Y2=Temp;
                    break ;



        --Level;
  Turtle_R=sqrt((X2-X1)* (X2-X1)+ (Y2-Y1)* (Y2-Y1))*LineLen;
                        XPoints[0]=X1;
        YPoints[0]=Y1;
        XPoints[NumLines]=X2;
        YPoints[NumLines]=Y2;
        Turtle_Theta=Point(X1,Y1,X2,Y2);
        Turtle_X=X1;
        Turtle_Y=Y1;
```

```
Turn(Angles[ 0 ]*Sign,Turtle_Theta);
    for (I=1; I<Split; ++I)
 -

        Step(Turtle_X, Turtle_Y, Turtle_R, Turtle_Theta);
            XPoints[ I ]=Turtle_X;
        YPoints[ I ]=Turtle_Y;
                Turn(Angles[ I ]*Sign,Turtle_Theta);

    for (I=NumLines -1; I>=NumLines -2; --I)
 -
                    Step(Turtle_X,      Turtle_Y,      Turtle_R,
Turtle_Theta);
                XPoints[ I ]=Turtle_X;
        YPoints[ I ]=Turtle_Y;
           Turn(Angles[ I ]*Sign,Turtle_Theta);

  Turtle_R=sqrt((XPoints[NumLines -2] - XPoints[Split -
  1]) * (XPoints[NumLines -2]- XPoints[Split -1]) +
    (YPoints[NumLines -2]- YPoints[Split -1])*
            (YPoints[NumLines -2]- YPoints[Split -
                    1]))*LineLen;
    Turtle_Theta= Point(XPoints[Split-1], YPoints[Split-
   1], XPoints[NumLines -2], YPoints[NumLines -2]);

  Turtle_X=XPoints [Split-1];
  Turtle_Y=YPoints [Split-1];
  Turn(-AngleSplit*Sign,Turtle_Theta);
for (I=Split ; I< 9 ;++I)
 -
                    Step(Turtle_X,      Turtle_Y,      Turtle_R,
Turtle_Theta);
                XPoints[ I ]=Turtle_X;
        YPoints[ I ]=Turtle_Y;
            Turn(Angles[ I ]*Sign,Turtle_Theta);


for (I=10; I>=9 ;--I)
 -
                    Step(Turtle_X,      Turtle_Y,      Turtle_R,
Turtle_Theta);
```

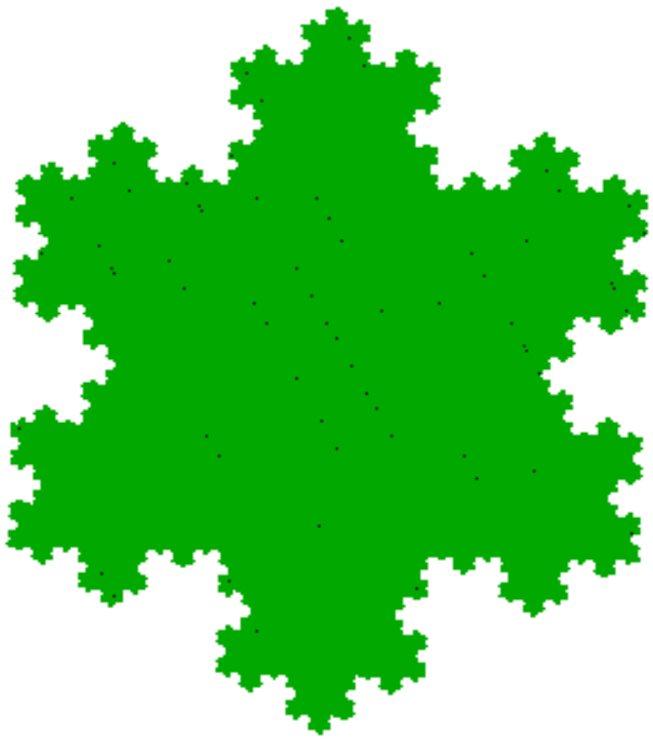
```
                    XPoints[ I ]=Turtle_X;
                YPoints[ I ]=Turtle_Y;
                    Turn(Angles[ I ]*Sign,Turtle_Theta);


            if(Level)
                for (I=0; I<NumLines; ++I)

                    switch(I)

                                                    case 1:
                            case 2:
                            case 3:
                            case 4:
                            case 8:
                            case 9:
                            case 12:
                                Type =0;
                                break;
                            case 0:
                            case 5:
                            case 6:
                            case 7:
                            case 10:
                            case 11:
                                Type = 1;
                                break;


                                X1 = XPoints[ I ];
                            Y1 = YPoints[ I ];
                        X2 = XPoints[ I+1 ];
                        Y2 = YPoints[ I+1 ];
        Peano13-DoanGenerator(pDC,X1,Y1,X2,Y2,Level,Type,Sign,
                                NumLines,LineLen,Angles);


            else
                        for (I= 0; I<NumLines; I++ )

                pDC->MoveTo((int)XPoints[ I ],(int) YPoints [ I ]);
                pDC->LineTo((int)XPoints[ I+1 ],(int) YPoints [ I+1

]);
```

```
delete[]XPoints;
delete[]YPoints;
```

Đối với đường này cũng có 4 khả năng lựa chọn cho vị trí của generator và phải chọn một cách cẩn thận đối với mỗi mức, mỗi đoạn thẳng để đảm bảo rằng đường cong tạo thành không tự giao nhau hay tự chồng lên nhau. Ở đây NumLines = 13 và mảng Angle là:

60,0,–60,–60,–60,0,60,60,60,0,60,0,–60

Tuỳ vào mức khác nhau thì tương ứng với hình vẽ khác nhau. Sau đây là hình minh hoạ của đường Peano 13-đoạn có mức là 5:



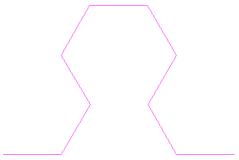
II.3 ĐƯỜNG SIERPINSKI:

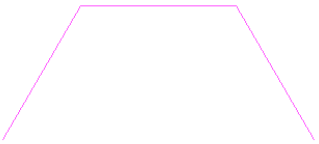
Đường Sierpinski được trình bày sau đây là một đường cong rất đặc biệt, bởi vì có rất nhiều cách phát sinh ra nó với các khởi động ban đầu hoàn toàn khác nhau nhưng lại kết thúc ở việc sinh ra một loại đường cong duy nhất.

Chúng ta đã quen với phương pháp đầu tiên để phát sinh ra tam giác Sierpinski bằng cách sử dụng kỹ thuật initiator / generator được mô tả ở các phần trước. Đối với đường này, initiator là một đoạn thẳng.

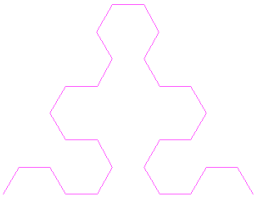
Generator đối với đường cong này và các đường được sinh ra ở mức 1, 2 và 3 được minh hoạ như sau:

Hình : Generator của tam giác Sierpinski m ức 2.



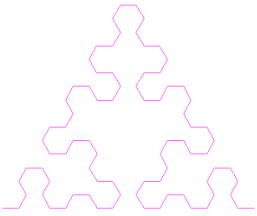


Generator của tam giác Sierpinski Mức 1

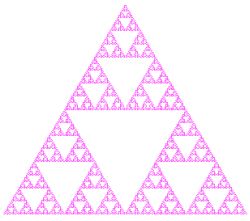


Mức 3

Và đây là đường Sierpinski ở mức 4 và 8:



Mức 4



Mức 8

Để phát sinh ra đường này ta dùng kỹ thuật giống như các đường họ Von Kock và Peano.

Đoạn mã của hàm Generator như sau:

```
 void Generator_SierpinskiCurve(CDC *pDC, double X1, double Y1,
                                double X2, double Y2, int Level,
                                int Sign, int NumLines, doubleLinelen,
                                double Angles[], COLORREF color)
{
 CPen pen;
 pen.CreatePen(PS_SOLID,1,color);
 CPen* pOldPen=pDC->SelectObject(&pen);
 double *XPoints,*YPoints;
 int i;
 int Init_Sign;
 double Turtle_Theta,TurtleX,TurtleY,TurtleR;
 XPoints=new double[NumLines+1];
 YPoints=new double[NumLines+1];
 TurtleR=sqrt((X2-X1)*(X2-X1)+(Y2-Y1)*(Y2-Y1))*Linelen;
 XPoints[0]=X1;
 YPoints[0]=Y1;
 XPoints[NumLines]=X2;
 YPoints[NumLines]=Y2;
 Turtle_Theta=Point(X1,Y1,X2,Y2);
 TurtleX=X1;
 TurtleY=Y1;
 Turn(Angles[0]*Sign,Turtle_Theta);
 for(i=1;i<NumLines;i++)
 {
        Step(TurtleX,TurtleY,TurtleR,Turtle_Theta);
        XPoints[i]=TurtleX;
        YPoints[i]=TurtleY;
        Turn(Angles[i]*Sign,Turtle_Theta);
 }
 --Level;
 Sign*=-1;
 if(Level)
 {
        Init_Sign=Sign;
        for(i=0; i<NumLines; i++)
        {
                X1=XPoints[i];
                Y1=YPoints[i];
                X2=XPoints[i+1];
                Y2=YPoints[i+1];
            Generator_SierpinskiCurve(pDC, X1, Y1, X2, Y2, Level,
                        Init_Sign, NumLines, Linelen, Angles, color);
                Init_Sign*=-1;
        }
 }
```

```
    else
        for(i=0;i<NumLines;i++)
        {
            pDC->MoveTo((int)XPoints[i],(int)YPoints[i]);
            pDC->LineTo((int)XPoints[i+1],(int)YPoints[i+1]);
        }
    pDC->SelectObject(pOldPen);
    delete []XPoints;
    delete []YPoints;
}
```

Với Num-line = 3 và mảng Angle là [60, -60, 0].

II.4 CÂY FRACTAL:

Trong các phần trước, chúng ta đã tạo ra các đường fractal bằng cách thay thế một cách lặp lại của các đoạn thẳng với các mẫu thu nhỏ của một generator mẫu, kết quả là các đường có tính tự đồng dạng. Bây giờ, chúng ta sẽ tạo ra đường cong theo một hướng khác. Chúng ta sẽ bắt đầu với một thân cây tại đầu mút của nó chúng ta tách thân cây thành hai hướng và vẽ hai nhánh. Chúng ta sẽ lặp lại quá trình này tại các đầu mút của mỗi nhánh. Kết quả chúng ta sẽ được một cây. Trước khi chúng ta biểu diễn các cây tự nhiên, đầu tiên chúng ta thảo luận vài điều về các cây thực tế.

□ CÁC CÂY THỰC TẾ:

Chúng ta phác thảo quá trình tạo cây được cho ở trên. Tại mỗi nút trong quá trình tạo cây, chúng ta tách làm hai hướng. Kết quả ta được một cây hai chiều. Chúng ta hy vọng nó có một số quan hệ với cây thực tế 3 chiều. Trước khi đi xa hơn, chúng ta quan sát một vài cây tự nhiên. Đầu tiên, có hai lớp cây là lớp cây rụng lá (deciduous) mỗi năm và lớp cây tùng bách (conifers). Hai lớp cây này hoàn toàn khác nhau. Cây tùng bách có khuynh hướng có các vòng của các nhánh ở tại các độ cao khác nhau vòng quanh trung tâm của thân cây. Điều này dường như không thích hợp với tất cả các quá trình rẽ nhánh nhị phân và chúng ta sẽ thấy các cây sau đây do chúng phát sinh không bao giờ giống với cây tùng bách thật sự.

Thứ hai, đối với cây rụng lá mặc dù sự xuất hiện của chúng rất gần với mô hình của chúng ta, thế nhưng vẫn còn rất nhiều phức tạp trong cấu trúc của chúng. Trong khi đó, việc rẽ nhánh nhị phân thường có qui luật và đơn giản hơn nhiều, chỉ ngoại trừ một vài thân cây có khả năng tách ra nhiều hơn hai nhánh.

□ BIỂN DIỄN TOÁN HỌC CỦA CÂY:

Theo Leonardo da Vinci quan sát, kết quả đó là do tổng số các vùng cắt ngang của các nhánh cây ở một độ cao cho trước là hằng số. Điều này không

gây ngạc nhiên vì cây đòi hỏi chuyển dinh dưỡng từ gốc đến lá và cho trước một lượng dinh dưỡng, một người nghĩ rằng thiết diện cần thiết cho sự vận chuyển sẽ không đổi bất kể chiều cao hay số ống dẫn. Khi chúng ta chuyển sự quan sát này vào các đường kính (hay các chiều rộng khi chúng ta vẽ thành cây hai chiều) thì chúng ta có được biểu thức sau:

Ở đây D_0, D_1, D_2 là đường kính của hai nhánh chia cây làm đôi, $\alpha = 2$ theo da Vinci. Do đó các dạng các dạng cấu trúc giống cây, mô hình đơn giản được cho ở trên có khả năng áp dụng cho các hệ thống sông tốt hơn các cây, vì thường có nhiều hơn hai con sông nhánh của một hệ thống sông sẽ nối với nhau ở cùng một nơi. Các cây khác được tìm thấy trong cơ thể con người là hệ thống động mạch và cuống phổi dùng để vận chuyển máu và oxy, trong đó α đối với hệ thống cuống phổi là 3 và đối với động mạch là 2.7.

Khi chúng ta xây dựng cây chúng ta sẽ sử dụng biểu thức:

$$B_{n+1} = 2^{\frac{-1}{\alpha}} B_n \qquad \text{(a)}$$

Ở đây B_n là đường kính của nhánh ở mức thấp hơn. B_{n+1} biểu diễn đường kính mỗi nhánh con khi B_n tách thành hai nhánh.

Chúng ta cũng cần xem xét chiều dài mỗi nhánh. McMahon nghiên cứu các loại cây kiểu mẫu khác nhau và đưa ra công thức như sau cho chiều dài:

$$L_{n+1} = 2^{\frac{-2}{3\alpha}} L_n \qquad \text{(b)}$$

Với L_n là chiều dài của nhánh trước đó và L_{n+1} chiều dài của mỗi nhánh trong hai nhánh kế sau khi nhánh trước đó được tách ra làm hai.

Để tạo thành một cây, ở đây chúng ta sử dụng đồ hoạ con rùa.

Gọi:
(X,Y) là toạ độ của gốc cây.
Height, Width là chiều cao và chiều rộng của cây.
Letf_Alpha, Right_Alpha là góc Alpha bên trái và góc Alpha bên phải.
Left_Angle, Right_Angle là góc rẽ bên trái và góc rẽ bên phải của nhánh.
Level là mức của cây.
Color1 là màu của thân cây.
Color2 là màu của tước cây.
Color3 là màu của lá cây.

Thuật toán:

(i) Tính các hệ số:

+ Chiều rộng trái và phải theo công thức (a).

```
Left_Width_Factor = pow (2, -1.0 / Left_Alpha );
Right_Width_Factor = pow (2, -1.0 / Right_Anpha );
```

+ Chiều cao trái và phải theo công thức (b)

```
Left_Height_Factor = pow (2, -2.0 / (3 * Left_Alpha));
Right_Height_Factor = pow (2, -2.0 / (3 * Right_Alpha));
```

(ii) Xác định toạ độ ngọn của thân cây:

```
X1 = X;
Y1 = Y + Height;
```

(iii) Vẽ thân cây từ (X, Y) đến (X1, Y1) với màu Color1 và chiều rộng là Width.

```
DrawLine (X, Y, X1,Y1, Color1, Width);
```

(iv) Phát sinh nhánh bên trái:

a) Xác định gốc giữa thân cây và trục x (tức là góc của con rùa)

```
Turtle_Theta = Point (X, Y, X1, Y1);
```

b) Quay con rùa về phía bên trái một góc Left

```
Turn (Left_Angle, &Turtle_Theta);
```

c) Sau đó gọi hàm Generator để phát sinh ra nhánh bên trái.

```
Generator (X1, Y1,Left_Width_Factor * Width,
Left_Height_Factor * Height, Level);
```

v) Phát sinh bên nhánh bên phải:

a) Xác định gốc giữa thân cây và trục x (tức là góc của con rùa)

```
Turtle_Theta = Point (X, Y, X1, Y1);
```

b) Quay con rùa về phía phải một góc Right_Angle

```
Turn (-Right_Angle, &Turtle_Theta);
```

c) Sau đó gọi hàm Generator để phát sinh ra nhánh bên phải

```
Generator (X1, Y1, Right_Width_Factor * Width,
        Right_Height_Factor * Height, Level);
```

Hàm Generator có đoạn mã như sau:

```
Generator (float X, float Y,float Width, float Height, unsigned char Level)
{
    (i) Xác định vị trí con rùa hiện tại và chiều dài một bước của con rùa
        Turtle_X = X;
        Turtle_Y = Y;
        Turtle_R = Height;
    (ii) Xác định ngọng của tước mới phát sinh và giảm mức đi một đơn vị.
        Step (&Turtle_X, &Turtle_Y, Turtle_R, Turtle_Theta);
```

```
                X2 = Turtle_X;
                Y2 = Turtle_Y;
                Level--;
```

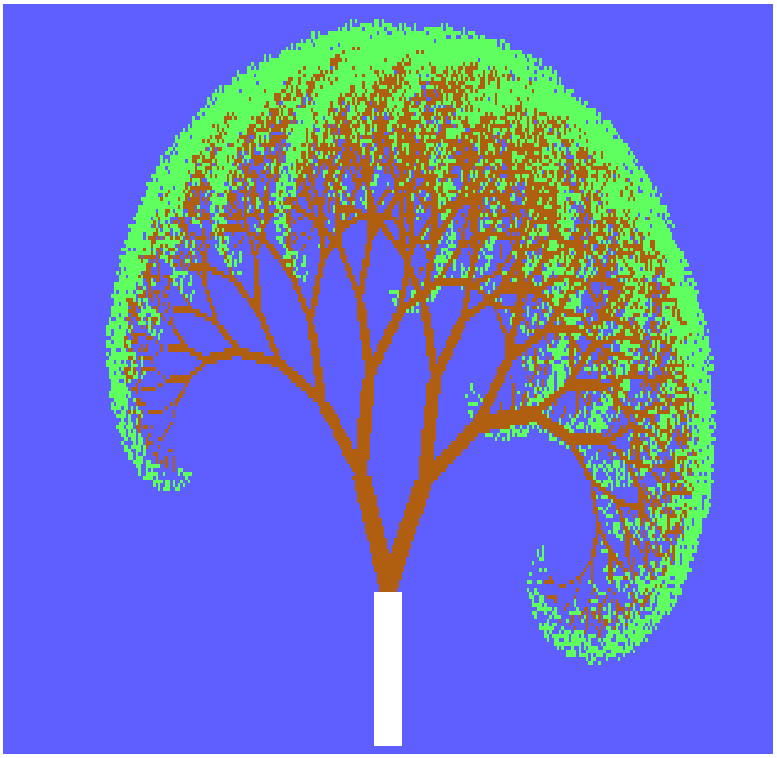
(iii) Vẽ đoạn thẳng từ (X, Y) đến (X2, Y2) với độ rộng là Width và màu được xác định như sau:

```
                + Nếu Level < 3 thì màu hiện thời là Color2.
                + Nếu Level >= 3 thì màu hiện thời là Color3.
                If (Level < 3)
                        DrawLine (X, Y, X2, Y2, Width, Color2);
                Else
                        DrawLine (X, Y, X2, Y2, Width, Color3);
```

iv) Nếu Level > 0 thì chúng ta tiếp tục phân làm hai nhánh trái và phải.

```
            if (Level > 0)
            {
                Turtle_Theta = Point(X, Y, X2, Y2);
                Turtle (Left_Angle, &Turtle_Theta);
        Generator (Turtle_X, Turtle_Y, Left_Width_Factor * Width,
                    Left_Height_Factor * Height, Level )
                Turntle_Theta = Point (X, Y, X2, Y2);
                Turn (- Right_Angle, &Turtle_Theta);
        Generator (X2, Y2, Right_Width_Factor * Width,
                    Right_Height_Factor * Height, Level);

            }
    }
```

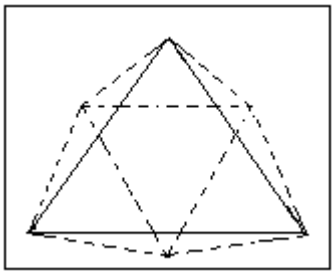
Sau đây là hình minh hoạ một cây fractal với Level = 14, Height = 80, Width = 20, Left_Alpha = 2.0, Right_Alpha = 2.2, Left_Angle = 20, Right_Angle = 28.



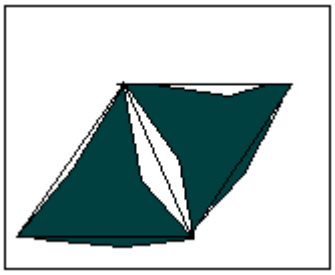
II.5 PHONG CẢNH FRACTAL:

Trong phần này chúng ta sẽ tạo ra phong cảnh fractal bằng cách sử dụng thay thế trung điểm.

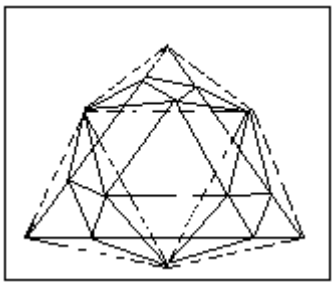
Các hình vẽ sau cho chúng ta thấy những bước đầu tiên trong quá trình thay thế trung điểm:



Hình (a)



Hình (b)



Hình (c)

Chúng ta bắt đầu bằng một tam giác và tiến hành thay thế trung điểm ứng với mỗi cạnh của tam giác này bằng một điểm trên đường trung trực của cạnh tương ứng. Khoảng cách giữa trung điểm cũ và trung điểm mới trong mỗi lần thay thế được xác định bởi việc nhân 1 hệ số ngẫu nhiên Gauss với độ dài đoạn thẳng. Kế tiếp chúng ta nối mỗi điểm vừa được tạo ra với hai đỉnh gần nhất của tam giác. Sau đó, từng cặp điểm mới tạo thành sẽ được nối lại với nhau. Cuối cùng chúng ta bỏ đi các cạnh của tam giác ban đầu.

Kết quả của quá trình này là sự thay thế tam giác ban đầu bằng 4 tam giác mới. Sau đó, chúng ta lại áp dụng quá trình xử lý trên mỗi một trong 4 tam giác mới này, lúc đó ta sẽ thu được từ 4 tam giác con 18 tam giác như hình vẽ (c).

Điều gì sẽ xảy ra khi chúng ta có hai tam giác có chung một cạnh, ngay cả khi bắt đầu với tam giác đơn, trạng thái này vẫn xuất hiện sau bước đầu tiên. Lúc đó chúng ta có cách giải quyết như sau:

Sự thay thế của cạnh chung đối với tam giác đầu tiên được hướng về phía bên trong tam giác này, còn sự thay thế của cạnh chung đối với tam giác thứ hai được hướng về phía bên trong trong tam giác đó. Nếu chúng ta giả sử đây là mức thấp nhất và lấp đầy các tam giác kết quả với một màu nào đó, thì

hiển nhiên mhận thấy rằng có một lổ hở không được lấp đầy. Có hai cách để giải quyết vấn đề này như sau:

Cách thứ nhất:

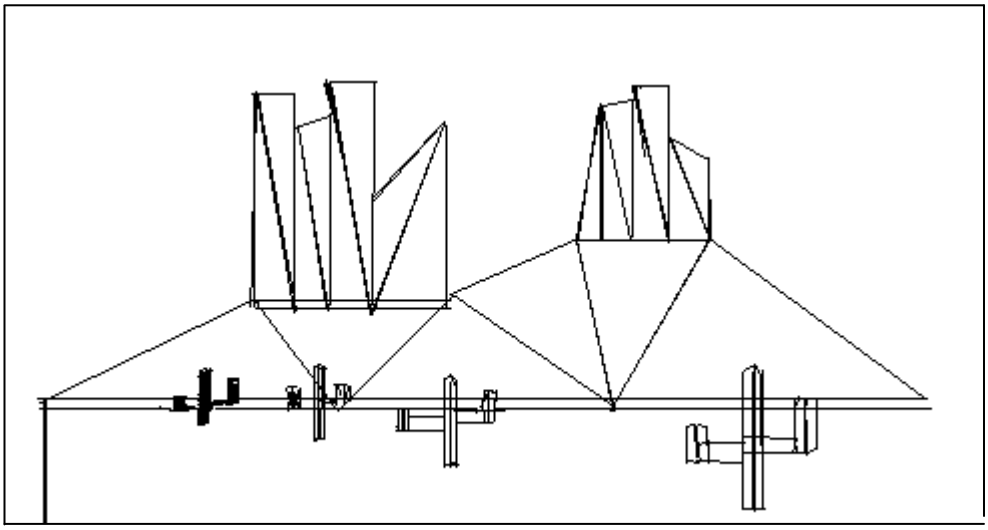
Cách này do Michael Batty đưa ra. Ở mỗi mức của quá trình đệ qui, ngoại trừ mức thấp nhất ông ta đã tạo ra một tam giác nhỏ hơn tam giác ban đầu và tô nó bằng một màu. Hy vọng rằng tam giác này đủ nhỏ sao cho nó không che các mặt không đều theo ý muốn được tạo bởi quá trình thay thế trung điểm nhưng nó đủ lớn sao cho nó tô được miền mà lổ hỏng sẽ phải xảy ra ở mức kế tiếp trở xuống của quá trình.

Cách thứ hai:

Sử dụng các toạ độ của trung điểm không được thay thế của đường để tạo ra một số duy nhất. Số này được sử dụng như hạt giống cho bộ phát sinh số ngẫu nhiên của máy tính để từ đó phát sinh ra các số ngẫu nhiên cho sự thay thế dọc theo trung trực của đoạn thẳng tương ứng. Khi đường trung trực này xuất hiện trong tam giác khác, vì trung điểm không được thay thế vẫn có cùng toạ độ hạt giống cho bộ phát sinh số ngẫu nhiên sẽ giống nhau và sự thay thế sẽ giống nhau, vì thế không có khả năng xảy ra lổ hỏng.

Chương trình sau tạo ra phong cảnh cây xương rồng trong hẻm núi đá gần Sedona, Arizona.

Để tạo được cảnh như thế, chúng ta dựa vào hình sau:



Khuôn cảnh trên để tạo nên hẻm núi đá ở Sedona, Arizano
Gọi toạ độ cửa sổ thực XWMin, YWMin, XWMax, YWMax

Sau đây là đoạn mã tạo ra phong cảnh này:

```
        Y_Max = 280;
double Level[22]=( 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4);
double X1[22]={-330, -90, -90, 120, 120, 120, -160, -120, -120, -80, -80,
    -50, -50,-50 , 80, 104, 104, 128, 128, 152, 152, 200};
```

```
double Y1[22]={-110, -110, -110, -110, -110, -110, -10, -10, -10, -10, -10,
      -10, -10,-10 , 50, 50, 50, 50, 50, 50, 50, 50};
double X2[22]={-160, -160, 0, 0, 80, 200, -160, -160, -120, -120, -80, -80,
      -50, 0,100, 100, 104, 104, 128, 128, 152, 152};
 double Y2[22]={0, 0, 0, 0, 50, 50, 220, 220, 190, 190, 230, 230,100, 180,
                  180,180,200, 205,215, 215, 160, 160 };
 double X3[22]={-90, 0, 120, 80, 200, 340, -120, -120, -80, -80, -50, -50, 0,
                  0, 104,104, 128, 128, 152, 152, 200, 200};
 double Y3[22]={-110, 0, -110, 50, 50, -110, -10, 220, -10, 200, -10, 235,
                180, -10, 50, 200, 50, 210, 50, 220, 50, 140};
    for(I=0 ; I<22 ; ++I)
    Generate(X1[ I ],Y1[ I ], X2[ I ],Y2[ I ], X3[ I ],Y3[ I ],Level[ I ],4,12);
    Y_Max= -100;
    Gen_Quad(-330, -260, -330, -100, 330, -100, 330, -260, 4, 6,14);
    Cactus(-110, -130, 3, 4, 2, 10);
    Cactus(-200, -120, 2, 4, 2, 10);
    Cactus(0, -160, 4, 4, 2, 10);
    Cactus(210, -200, 6, 4,2,10);
```

Trong đó các hàm có đoạn mã sau:

```
    void Node(double X1, double Y1, double X2, double Y2, double X3,
              double Y3, double X4,  double Y4, double X5, double Y5,
              double X6, double Y6, unsigned char Level, int Color1,
              int Color2)
    {
        if(!Level)
            return;
        Generate(X1,Y1, X4,Y4, X6,Y6,Level-1,Color1,Color2);
        Generate(X6,Y6, X5,Y5, X3,Y3,Level-1,Color1,Color2);
        Generate(X4,Y4, X2,Y2, X5,Y5,Level-1,Color1,Color2);
        Generate(X4,Y4, X5,Y5, X6,Y6,Level-1,Color1,Color2);
    }


    void Generate(double X1, double Y1, double X2, double Y2, double
                  X3, double Y3, unsigned char Level, int Color1, int
                  Color2)
    {

      double X4,Y4,X5,Y5,X6,Y6,Ax,Ay,Bx,By,Cx,Cy;

                  X=X2-X1;
                  Y=Y2-Y1;
                   MidPoint();
                   X4=X1+Xz-Xp;
```

```
                Y4=Y1+Yz-Yp;
                Ax=-Xp;
                Ay=-Yp;

        X=X3-X1;
                 Y=Y3-Y1;
                 MidPoint();
                 X6=X1+Xz;
               Y6=Y1+Yz;
               Cx=Xp;
               Cy=Yp;

        X=X3-X2;
                 Y=Y3-Y2;
                 MidPoint();
                 X5=X2+Xz;
              Y5=Y2+Yz;
              Bx=-Xp;
              By=-Yp;
        if(Level)
        {
    PlotTriange(X1,Y1,X4+Ax,Y4+Ay,X6+Cx,Y6+Cy,Color1,Color2);
    PlotTriange(X6+Cx,Y6+Cy,X5+Bx,Y5+By,X3,Y3,Color1,Color2);
    PlotTriange(X4+Ax,Y4+Ay,X5+Bx,Y5+By,X6+Cx,Y6+Cy,Color1,
                                                        Color2);
    PlotTriange(X4+Ax,Y4+Ay,X2,Y2,X5+Bx,Y5+By,Color1,Color2);
    Node(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X6,Y6,Level, Color1,
                                                        Color2);
        }
        else
        {

           PlotTriange(X1,Y1,X4,Y4,X6,Y6,Color1,Color2);
           PlotTriange(X6,Y6,X5,Y5,X3,Y3,Color1,Color2);
           PlotTriange(X4,Y4,X5,Y5,X6,Y6,Color1,Color2);
           PlotTriange(X4,Y4,X2,Y2,X5,Y5,Color1,Color2);
        }
}

void PlotTriange(double X1, double Y1, double X2, double Y2, double
                    X3,  double Y3, int Color1, int Color2)
{
        int Color;
        double C1=0.35;
        double C2=0.92;
        double Ytt,Zt;
```

```
        Ytt=(Y1>Y2)?Y1:Y2;
        if(Ytt<Y3)
                Ytt=Y3;
        Zt=(Y_Max+YWMax)*(1-(Ytt+YWMax)/(Y_Max+YWMax) *
            (Ytt+YWMax)/(Y_Max+YWMax));
        if(random(Y_Max+YWMax+1)<=Zt)
                Color=Color1;
        else
                Color=Color2;
        if((Ytt+YWMax) < (C1*(Y_Max+YWMax)))
                Color=Color1;
        if((Ytt+YWMax) > (C2*(Y_Max+YWMax)))
                Color=Color2;
        FillTriange(X1,Y1, X2,Y2, X3,Y3,Color); //Lấp đầy tam giác
        với màu Color
}
void MidPoint()
{
        double R,W,C=1.0/2;
        double Lstart1=0,Lend1=1.0/6;
        double Lstart2=0.03,Lend2=0.07;
        R=C+Random_No(LStart1,LEnd1);
        W=Random_No(LStart2,LEnd2);
        Xz=R*X-W*Y;
        Yz=R*Y-W*X;
        C=0.05;
        Xp=C*Y;
        Yp=-C*X;
}
double Random_No(double LimitStart,double LimitEnd)
{
        int Half=MAXINT/2;
        LimitEnd-=LimitStart;
        LimitEnd=Half/LimitEnd;
        Double Result=(rand() - half)/LimitEnd;
        if(Result >= 0)
                Result+=LimitStart;
        else
                Result-=LimitStart;
           Return Result;

}

void Gen_Quad(double X1, double Y1, double X2, double Y2, double
                    X3, double Y3, double X4, double Y4, unsigned char
                    Level, int Color1, int Color2)
```

```
{
        Generate(X1,Y1, X2,Y2, X3,Y3,Level,Color1,Color2);
        Generate(X1,Y1, X4,Y4, X3,Y3,Level,Color1,Color2);
}

void Cactus(double X1, double Y1, int Scale, unsigned char Level,
                int Color1, int Color2)

{
    Gen_Quad(X1, Y1, X1, Y1+21*Scale, X1+1.6*Scale, Y1+22*Scale,
                    X1+1.6*Scale,Y1,Level,Color1,Color2);
        Gen_Quad(X1+1.4*Scale, Y1, X1+1.4*Scale, Y1+22*Scale,
                    X1+3*Scale, Y1+21*Scale,X1+3*Scale,
                    Y1,Level,Color1,Color2);
        Gen_Quad(X1, Y1+9*Scale, X1+7*Scale, Y1+9*Scale,
                    X1+7*Scale,Y1+12*Scale,X1, Y1+12*Scale, 0,
                    Color1,Color2);
        Gen_Quad(X1, Y1+9*Scale, X1+6*Scale, Y1+9*Scale,
                    X1+7*Scale,Y1+12*Scale,X1, Y1+12*Scale,
                    Level,Color1,Color2);
        Gen_Quad(X1+7*Scale, Y1+9*Scale, X1+7*Scale,
                    Y1+16*Scale, X1+8.5*Scale, Y1+17*Scale,
                    X1+8.5*Scale, Y1+9*Scale, Level, Color1, Color2);
        Gen_Quad(X1+8.4*Scale, Y1+9*Scale, X1+8.4*Scale,
                    Y1+16*Scale,X1+10*Scale, Y1+17*Scale,
                    X1+10*Scale,Y1+10*Scale, Level, Color1,Color2);
        Gen_Quad(X1, Y1+7*Scale, X1-6*Scale, Y1+7*Scale,
                    X1 - 6*Scale, Y1+10*Scale, X1,   Y1+10*Scale,0,
                    Color1,Color2);
        Gen_Quad(X1, Y1+7*Scale, X1-6*Scale, Y1+7*Scale,
                    X1 -6*Scale,Y1+10*Scale, X1,Y1+10*Scale,
                    Level,Color1,Color2);
        Gen_Quad(X1-7*Scale, Y1+8*Scale, X1-7*Scale, Y1+12*Scale,
                    X1+5.4*Scale, Y1+13*Scale, X1+5.4*Scale,
                    Y1+7*Scale, Level,Color1,Color2);
        Gen_Quad(X1-5.6*Scale, Y1+7*Scale, X1-5.6*Scale,
                    Y1+13*Scale, X1-4*Scale, Y1+12*Scale, X1-
                    4*Scale, Y+7*Scale, Level,Color1,Color2);

}
```

Để vẽ phong cảnh này, chúng ta sử dụng kỹ thuật lấp đầy tam giác được chia nhỏ của Michael Batty ở các giai đoạn trung gian nhằm tránh các lổ hỏng.

Đầu tiên, chúng ta xem qua hàm Generator. Hàm này xác định chiều dài theo hướng x và y cho mỗi đoạn của tam giác có toạ độ (X1, Y1), (X2, Y2),

(X3,Y3), sau đó gọi hàm MidPoint để xác định các phép thay thế trung điểm theo hướng x và y. Toạ độ của trung điểm thay thế được lưu trữ và các phép thay thế cần xác định một tam giác được chia nhỏ và lấp đầy ở mức trên mức thấp nhất, các đỉnh của tam giác này được lưu trữ trong các vị trí Ax, Ay, Bx, By, Cx, Cy. Nếu chúng ta ở mức thấp nhất (mức 0), hàm này gọi hàm PlotTriange để xác định màu lấp đầy và thực hiện việc lấp đầy 1 trong 4 tam giác mới, nếu mức thấp nhất chưa đạt đến, nó sẽ gọi hàm PlotTriange để lấp đầy 1 trong 4 tam giác được chia nhỏ và sau đó gọi hàm Node, hàm này gọi đệ quy hàm Generator để phát sinh ra 4 tam giác mới từ 1 trong 4 tam giác vừa được tạo ra.

Đối với hàm Node, nếu Level = 0 thì thoát, còn đối với các trường hợp khác thì nó gọi hàm Generator cho lần lượt từng tam giác trong 4 tam giác vừa được tạo thành.

Hàm Gen_Quad chỉ chạy Generator đối với hai tam giác tạo thành một hình thang.

Hàm Random_No được sử dụng trong việc xác định thay thế ngẫu nhiên, hàm có 2 tham số giới hạn trên và dưới (Cả 2 giá trị này đều là số dương) của số ngẫu nhiên được phát sinh. Số ngẫu nhiên trả về sẽ là số âm nằm giữa hai giá trị âm của hai số giới hạn hoặc dương nằm giữa hai giá trị dương của hai số giới hạn.

Còn hàm MidPoint ban đầu lấy số ngẫu nhiên và được chọn biểu diễn cho việc thay thế trung điểm dọc theo đường trung trực với khoảng cách theo chiều x được lưu trữ trong giá trị X và khoảng cách theo chiều y được lưu trữ trong giá trị Y. Khoảng cách này bằng nửa độ dài cạnh ứng với trung trực cộng hay trừ với một giá trị ngẫu nhiên giữa 0 và 1/6 lần chiều dài cạnh đó. Kế đến chúng ta tính độ dịch chuyển vuông góc với cạnh này. Nó bằng độ dài cạnh đang xét nhân với một số ngẫu nhiên giữa 0.03 và 0.07 hay giữa -0.07 và 0.03.

Hàm PlotTriange có các tham số là 3 đỉnh của tam giác và hai giá trị màu, nó dùng biến Y_Max là giá trị độ cao điều khiển việc chọn lựa màu. Đầu tiên hàm này chọn giá trị y của đỉnh cao nhất trong tam giác. Sau đó nó tạo biến Zt theo công thức:

$$Zt = (Y\_Max + YWMax)\left(1 - \left(\frac{Ytt + YWMax}{Y\_Max + YWMax}\right)^2\right)$$

Với Y_Max là độ cao điều khiển và Ytt là độ cao của đỉnh cao nhất của tam giác.

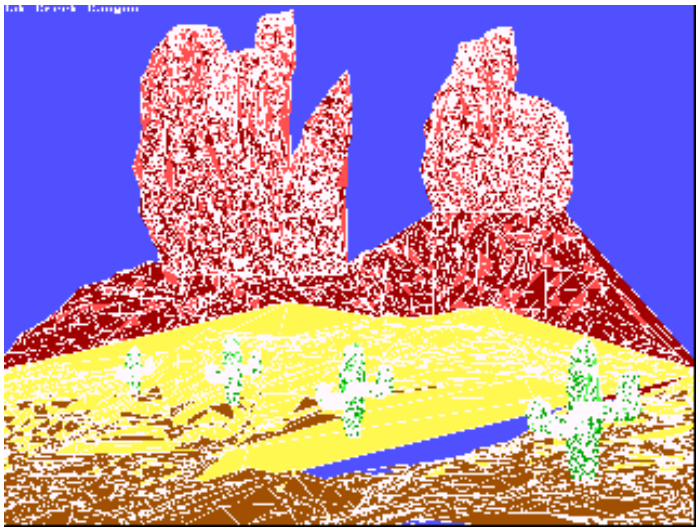
Khi giá trị Zt đã được xác định, hàm PlotTriange sẽ chọn một số ngẫu nhiên giữa 0 và Y_Max rồi so sánh giá trị này với Zt. Nếu giá trị này nhỏ hơn

hay bằng Zt, màu thứ nhất sẽ được chọn, ngược lại màu thứ hai được chọn. Cuối cùng nếu độ cao Ytt dưới giới hạn được chọn thì màu được chọn là màu thứ nhất, ngược lại chọn màu thứ hai. Sau đó hàm này gọi hàm FillTriange để lấp đầy tam giác với màu được chọn.

Hàm Cactus có các tham số là các toạ độ, hệ số vị tự, mức và hai màu. Nhiệm vụ của nó là phát sinh ra các cây xương rồng.

Đoạn mã chạy phong cảnh ở trên bắt đầu là chạy vòng for để gọi hàm Generator 22 lần để tạo ra vách đá màu đỏ. Sau đó gọi hàm Gen_Quad để vẽ nền sa mạc màu vàng và màu nâu, cuối cùng nó gọi hàm Cactus bốn lần để tạo 4 cây xương rồng với các vị trí và kích thước khác nhau.

Bức tranh hẻm núi đá được thực hiện bằng kỹ thuật tạo phong cảnh fractal.



II.6 HỆ THỐNG HÀM LẶP (IFS)

□ CÁC PHÉP BIẾN ĐỔI AFFINE TRONG KHÔNG GIAN R^2

Cho phép biến đổi w: $IR^2 \rightarrow IR^2$ có dạng:

w (x, y) = (ax + by + e, cx + dy + f)

Ở đây a, b, c, d, e, f là các hệ số thực, được gọi là phép biến đổi affine (hai chiều).

Phép biến đổi có thể viết dưới dạng:

$$w\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} = AX + T$$

Với:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, X = \begin{pmatrix} x \\ y \end{pmatrix}, T = \begin{pmatrix} e \\ f \end{pmatrix}$$

Bằng cách biểu diễn phép biến đổi trên dưới dạng tách các phép quay và vị tự ma trận A có thể viết dưới dạng:

$$A = \begin{pmatrix} r\cos\theta & -s\sin\phi \\ r\sin\theta & s\cos\phi \end{pmatrix}$$

Với:

r: hệ số vị tự trên trục x.
s: hệ số vị tự trên trục y.
θ: góc quay trên trục x.
ϕ: góc quay trên trục y.
e: hệ số tịnh tiến trên trục x.
f: hệ số tịnh tiến trên trục y.

□ IFS CỦA CÁC PHÉP BIẾN ĐỔI AFFINE TRONG KHÔNG GIAN R^2:

Một IFS là tập hợp các phép biến đổi affine co tức là:
IFS { IR^2 ; w_n : n = 1, 2,…, N } với w_n là phép biến đổi affine.

Ví dụ:
Một IFS của ba phép biến đổi w_1, w_2, w_3 là IFS { IR^2 ; w_1, w_2, w_3 } với w_1, w_2, w_3 xác định bởi:

$$w_1\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$w_2\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$w_3\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.25 \\ 0.25 \end{pmatrix}$$

Trong đó mỗi phép biến đổi affine liên kết với một xác xuất P_n quyết định độ quan trọng của nó so với phép biến đổi khác. Để ý rằng:

$$\sum_{n=1}^{N} P_n = 1, P_n > 0 (n = 1,..., N)$$

Ví dụ:

Mã IFS đối với tam giác Sierpinski

w	*a*	*b*	*c*	*d*	*e*	*f*	*p*
1	0.5	0	0	0.5	0	0	0.33
2	0.5	0	0	0.5	1	0	0.33
3	0.5	0	0	0.5	0.5	0.5	0.34

□ GIẢI THUẬT LẶP NGẪU NHIÊN:

Cho IFS [IR^2 ; $\mathbf{w_n}$: n = 1, 2, …, N] , mỗi w_n liên kết với xác xuất P_n.

Trước khi trình bày thuật toán, chúng ta tìm cách chọn các xác xuất P_n thích hợp.

Khi chúng ta xác định các phép biến đổi, chúng ta cần chọn các xác xuất cho chúng. Việc chọn các xác xuất khác nhau sẽ không dẫn đến các hấp tử khác nhau, nhưng chúng ảnh hưởng đến tốc độ ở các miền khác nhau hay thuộc tính của hấp tử được làm đầy.

Mỗi phép biến đổi affine $\mathbf{w_n}$ tương ứng với hấp tử J là:

$$w_n\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}a_n & b_n\\c_n & d_n\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}+\begin{pmatrix}e_n\\f_n\end{pmatrix}, n=1,2,\ldots,N$$

Số lần mà điểm nhảy một cách ngẫu nhiên dùng trong hấp tử con $\mathbf{w_n}$ xấp xỉ với:

$$\frac{s(w_n)}{s(J)}$$

Với S(A): Diện tích của A

Nếu $\det A_n \neq 0$ thì việc chọn xác xuất P_n như sau:

$$P_n=\frac{|\det A_n|}{\sum_{i=1}^{N}|\det A_i|}=\frac{|a_nd_n-b_nc_n|}{\sum_{i=1}^{N}|a_id_i-b_ic_i|}$$

Nếu $\det A_n = 0$ thì P_n được gán cho một số nhỏ nhất và khá gần 0, ví dụ như 0.001. Sau đây là các bảng mã IFS:

✸ Mã IFS cho lá dương xỉ:

W	A	b	C	d	E	f	p
1	0	0	0	0.16	0	0	0.01
2	0.2	-0.26	0.23	0.22	0	1.6	0.07
3	-0.15	0.28	0.26	0.24	0	0.44	0.07
4	0.85	0.04	-0.04	0.85	0	1.6	0.85

Thuật toán lặp ngẫu nhiên:

(i) Khởi động các giá trị

x = 0;

y = 0;

(ii) for (i = 1; i < number_iterates; ++i)

{

+ Chọn k là một trong số 1, 2, …, N

+ Áp dụng phép biến đổi $\mathbf{w_k}$ cho điểm (x, y) ta thu được diểm (x~,y~).

+ Đặt (x, y) bằng điểm mới

x = x~;

y = y~;

+ putpixel(x, y, color);

}

Tương tự chúng ta áp dụng thuật toán này đối với IFS của phép biến đổi affine trong không gian ba chiều có dạng:

$$w\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & m \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} n \\ q \\ r \end{pmatrix} = \begin{pmatrix} ax+by+cz+n \\ dx+ey+fz+q \\ gx+hy+mz+r \end{pmatrix}$$

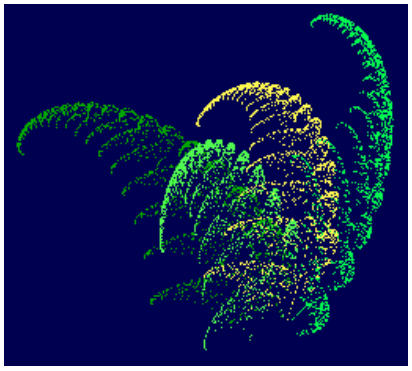
✸ Mã IFS cho lá dương xỉ ba chiều:

w	A	B	c	D	E	f	G	h	m	n	q	r	P
1	0	0	0	0	0.18	0	0	0	0	0	0	0	0.001
2	0.83	0	0	0	0.86	0.1	0	-0.12	0.84	0	1.62	0	0.901
3	0.22	-0.23	0	0.24	0.22	0	0	0	0.32	0	0.82	0	0.049
4	0.22	0.23	0	0.24	0.22	0	0	0	0.32	0	0.82	0	0.049

Sau đây là các hình vẽ minh hoạ giải thuật lặp ngẫu nhiên tương ứng với bảng mã IFS được trình bày ở phần trước:

Lá dương xỉ 2 chiều

Lá dương xỉ 3 chiều



II.7 TẬP MANDELBROT

□ Đặt vấn đề:

Trong nhiều thập niên của thế kỷ XX, các nhà toán học đã để tâm nghiên cứu đến một loại biểu thức phức tạp xác định bởi:

$z_{n+1} = z_n^2 + c$, trong đó $z_i \in C$, $\forall i \in N$ & $c \in C$ (1)

Để đơn giản hoá vấn đề, trước hết ta xét trường hợp $c = 0$ và $z_0 \in R$. Khi đó có 3 trường hợp sau:

+ $z_0 = 1$: khi đó $z_n = 1$, $\forall n \geq 1$.
+ $z_0 < 1$: khi đó $z_n \rightarrow 0$ khi $n \rightarrow \infty$.
+ $z_0 > 1$: khi đó $z_n \rightarrow \infty$ khi $n \rightarrow \infty$.

Ở đây tốc độ tiến đến 0 hay tiến đến ∞ của dãy (z_n) được quyết định bởi giá trị ban đầu z_0 của dãy. Trong trường hợp $z_0 < 1$, giá trị z_0 càng nhỏ thì dãy (z_n) tiến đến 0 càng nhanh. Ngược lại khi $z_0 > 1$, giá trị z_0 càng lớn thì dãy (z_n) càng tiến nhanh ra ∞.

Trong trường hợp tổng quát, dãy (z_n) được xác định bởi công thức (1) ở trên rất khó khảo sát về mặt lý thuyết. Chỉ đến năm 1979, Mandelbrot mới thành công trong việc quan sát dãy này với sự hỗ trợ của máy tính điện tử. Kết quả được Mandelbrot quan sát thấy là một trong những cấu trúc fractal phức tạp và đẹp. Nó đã được đặt tên Mandelbrot để ghi nhớ công lao của tác giả, người đã khai sinh ra lý thuyết hình học phân hình.

□ CÔNG THỨC TOÁN HỌC:

Ký hiệu $z_n = (x_n , y_n)$, $c = (p,q)$, trong đó:

$x_n = Re(z_n)$, $p = Re(c)$, $y_n = Im(z_n)$, $q = Im(c)$, $\forall n \geq 0$ thì hệ thức truy hồi xác định ở (1) có thể được viết lại theo dạng đơn giản hơn như sau:

$$x_{n+1} = x_n^2 - y_n^2 + p$$

$$y_{n+1} = 2x_n . y_n + q \qquad (2)$$

Ngoài ra khi khảo sát dãy (z_n) ta tìm được tính chất sau:

Tính chất về sự hội tụ của dãy (z_n):

- Nếu tồn tại $k \in N$ sao cho $| z_k | > 2$ thì dãy (z_n) hội tụ đến vô cực.
- Nếu tồn tại $k \in N$ sao cho $| z_t | < 2$, $\forall t : k \leq t \leq 1$, với 1 là hằng số hữu hạn thì cũng có $| z_n | < 2$, $\forall n \geq k$. (Ký hiệu $| z |$ chỉ modul của số phức z).

□ THUẬT TOÁN THỂ HIỆN TẬP MANDELBROT:

1. Xây dựng thuật toán:

Tập Mandelbrot là hình ảnh của dãy (z_n), với giá trị khởi đầu $z_0 = 0$. Khi đó màn hình máy tính sẽ chuyển đổi thành một mặt phẳng phức thu hẹp với:

+ Trục x biểu diễn phần thực của số phức c (giá trị p được nêu ở phần 2/).

+ Trục y biểu diễn phần ảo của số phức c (giá trị q được nêu ở phần 2/).

Từ tính chất về sự hội tụ của dãy (z_n) ở phần 2 chúng ta có thể chia tập các giá trị của c trên mặt phẳng phức thành 2 lớp:

Lớp 1:

Gồm các giá trị c làm cho dãy (z_n) không tiến ra vô cực mà được giới hạn trong một vòng tròn bán kính 2. Một cách cụ thể, đó là các giá trị c sao cho khi xuất phát từ chúng, ta luôn có $| z_i | < 2$, $\forall i = 1, 2, \ldots, l$, trong đó l do ta chọn trước. Để ý là giá trị l càng lớn thì tính hội tụ của dãy (z_n) tương ứng với một giá trị cụ thể càng được kiểm tra chặt chẽ và chính xác. Tuy nhiên khi đó thời gian tính toán để xác định tính hội tụ sẽ tăng lên gấp nhiều lần.

Lớp 2:

Gồm các giá trị phức c làm cho dãy (z_n) hội tụ về vô cực. Cụ thể đó là các giá trị c khởi đầu dẫn đến $| z_n | > 2$ ở một ngưỡng k hữu hạn nào đó.

Vấn đề đặt ra ở đây là cần quan sát tính hỗn độn của dãy (z_n). Do đó chúng ta tập trung các quan sát vào các giá trị c thuộc lớp 2. Muốn như vậy các

giá trị này phải được thực hiện một cách nổi bật trên màn hình máy tính bởi các màu khác nhau. Chúng ta sẽ tô màu mặt phẳng phức màn hình theo qui tắc sau:

+ Các giá trị c thuộc lớp 1 được tô màu đen vì không có tính chất gì đáng chú ý.

+ Các giá trị c thuộc lớp 2 được tô bằng các màu khác nhau ứng với các ngưỡng tiến ra vô hạn k khác nhau. Do số lượng màu có thể hiển thị trên một màn hình đồ hoạ là hữu hạn, việc tô màu các giá trị này sẽ được thực hiện theo kỹ thuật tô màu xoay vòng được chỉ ra ở các phần tiếp sau đây.

2. Thuật toán tổng quát để thể hiện tập Mandelbrot:

Thuật toán gồm các bước sau:

- Bước 1:

Xuất phát với một giá trị khởi đầu c = (p,q).

- Bước 2:

Kiểm tra c thuộc lớp 1 hay lớp 2.

- Bước 3:

Nếu c thuộc lớp 1 thì tô điểm ảnh tương ứng với c trên màn hình bằng màu đen, ngược lại tô điểm ảnh này bởi màu tương ứng xác định từ kỹ thuật tô xoay vòng.

- Bước 4:

Chọn một giá trị c mới và trở lại bước 1 cho đến khi quét hết toàn bộ giá trị c cần khảo sát (đôi khi chúng ta không cần khảo sát toàn bộ mà chỉ khảo sát một miền con được yêu cầu của mặt phẳng phức). Khi đó thuật toán kết thúc.

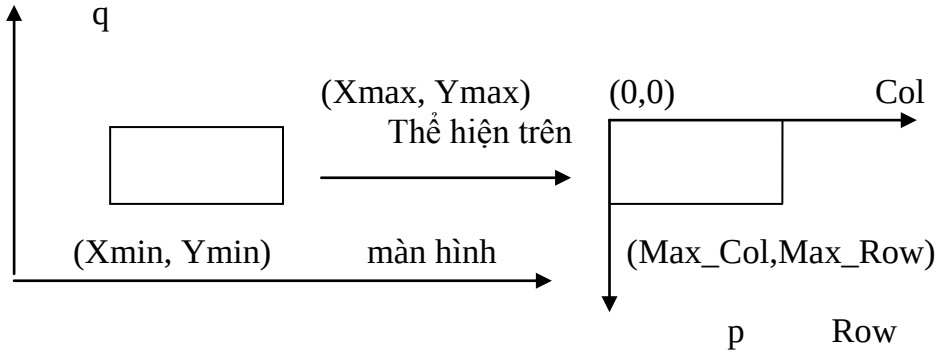
Bây giờ ta ký hiệu:

+ Max_Iterations là số lần lặp tối đa cần có để kiểm tra giá trị c thuộc lớp 1 hay lớp 2 (chính là giá trị 1 được đề cập trong định nghĩa của lớp 1).

+ Count là số lần lặp đã thực hiện (giá trị này tương ứng với một ngưỡng tiến ra vô hạn k được nêu ra trong định nghĩa lớp 2).

+ Miền con của mặt phẳng phức cần khảo sát là một cửa sổ hình chữ nhật được mô tả bởi toạ độ góc trái bên dưới (X_{min} , Y_{min}) và toạ độ góc phải trên (X_{max} , Y_{max}) (theo hệ trục toạ độ thông thường).

Khi đó mối liên hệ giữa hệ trục toạ độ phức thực tế với hệ toạ độ nguyên trên màn hình máy tính được x thể hiện bởi hình 11.1 dưới đây:



Vùng khảo sát là 1 miền con của mặt phẳng phức biểu biễn giá trị c

Hệ toạ độ nguyên trên máy tính với chiều dương ngược chiều thực tế.

Hình 11.1

Theo hình này, điểm bắt đầu (0, 0) trên màn hình máy tính sẽ tương ứng với giá trị c = (X_{min} , Y_{max}), điểm kết thúc (Max_Col, Max_Row), trong đó Max_Col x Max_Row thể hiện độ phân giải của mode đồ hoạ hiện thời của màn hình (ví dụ với mode VGA 16 màu ta có Max_Col = 604, Max_Row = 480), sẽ tương ứng với điểm c = (X_{max} , Y_{max}). Do đó nếu ký hiệu:

Δp là lượng gia tăng theo theo trục thực của giá trị p ứng với mỗi cột trên màn hình.

Δq là lượng gia tăng theo trục ảo của giá trị q ứng với mỗi hàng trên màn hình thì:

$$\Delta p = \frac{X_{\max} - X_{\min}}{Max\_Col}$$

$$\Delta q = \frac{Y_{\max} - Y_{\min}}{Max\_Row}$$

Khi đó một số phức c = (p, q) ứng với một điểm ảnh có toạ độ màn hình là (Col, Row) sẽ được xác định bởi:

$$p = X_{min} + Col.\Delta p$$
$$q = Y_{max} - Row.\Delta q$$

Có thể kiểm tra là khi Col, Row biến thiên theo chiều tăng đến các giá trị tương ứng Max_Col, Max_Row, chúng ta cũng có p biến thiên từ X_{min} đến X_{max} và q biến thiên từ Y_{max} xuống Y_{min} , đúng như yêu cầu về quan hệ giữa hệ

toạ độ phức sử dụng trong toán học với hệ toạ độ hiển thị của màn hình đã được nêu trong Hình 11.1.

Việc kiểm tra c thuộc lớp 1 hay 2 được viết dưới dạng:

if (Count > Max_Iterations) & (Modul (Z_{count}) < 2))

c thuộc lớp 1

if (Count < Max_Iterations) & (Modul (Z_{count}) > 2))

c thuộc lớp 2

Đồng thời màu tô cho một điểm c = (p, q) thuộc lớp 2 được tính toán theo công thức:

Màu tô (p, q) = Count(p, q) mod Max_Colors

Với:

Màu tô (p, q): số hiệu màu gán cho điểm ảnh tương ứng với giá trị (p, q)

Count(p, q): ngưỡng tiến ra vô hạn của dãy (z_n) tương ứng với (p, q).

Max_Colors: số lượng màu tối đa có trong palette màu đang được sử dụng.

Trong thuật toán này, để thể hiện một cách chi tiết, chúng ta quét các giá trị c trong cửa sổ giới hạn bởi (X_{min} , Y_{min}) và (X_{max} , Y_{max}) theo thứ tự từ trên xuống dưới và từ trái sang phải, tức là ứng với mỗi cột Col, ta kiểm tra và tô màu tất cả các điểm ảnh trên cột này theo các giá trị phức tương ứng, sau đó mới chuyển sang cột mới kế tiếp đó.

Như vậy chúng ta có được thuật toán chi tiết sau:

```
for(Col = 0; Col < Max_Col; ++Col)
 {
        for(Row = 0; Row <= Max_Row; ++Row)
          {
                  X = Y = x_n = y_n = 0;
  (vì ứng với mỗi giá trị c = (p, q) tương ứng với điểm ảnh
            (Col, Row), dãy (z_n) được khảo sát với z_0 = 0)
                  Count = 1;
                  While (Count < Max_Iterations) & (√ X + Y  < 2)
                   {
                         X = x_n^2 ;
                         Y = y_n^2 ;
                         y_n = 2x_n y_n + q;
                         x_n = X – Y + p;
                         Count = Count + 1;
                   }
                  Tô màu điểm ảnh (Col, Row) bởi màu có số hiệu Count
mod
                                  Max_Colors ;
                        q = q + Δq ;
          }
```

p = p + Δp ;

}

Chúng ta có nhận xét đầu tiên là trong thuật toán trên, giá trị q được tính tất cả Max_Col x Max_Row lần, như vậy với màn hình có độ phân giải 640 x 480, q được tính 307200 lần, nhưng thực chất chỉ có 480 giá trị q ứng với 480 hàng được sử dụng. Do đó để tăng tốc độ làm việc của thuật toán, các giá trị q có thể được tính trước khi vào vòng lặp và được lưu lại trong 1 mảng 1 chiều Q có Max_Row phần tử như sau:

```
Q[0] = Ymax ;
for(i = 0; i < Max_Row; ++i)
        Q[i] = Q[i - 1] - Δq;
```

Ở đây Δq được tính trước theo công thức đã nêu.

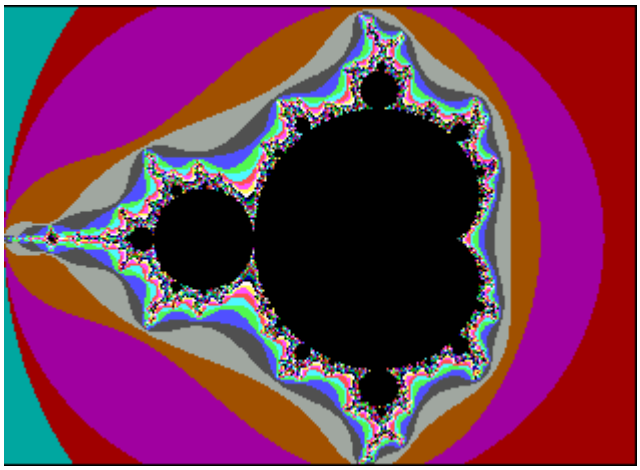
Một nhận xét thứ hai là việc tính $| z_n | = \sqrt{X + Y} = \sqrt{x_n^2 + y_n^2}$ để so sánh với 2 chiếm rất nhiều thời gian.

Do đó chúng ta thay việc so sánh $\sqrt{x_n^2 + y_n^2} < 2$ bởi $x_n^2 + y_n^2 < 4$ vì hai bất đẳng thức này tương đương. Việc làm này sẽ làm giảm đáng kể thời gian thực hiện thuật toán.

Thuật toán được viết lại dưới dạng:

```
for(Col= 0; Col<Max_Col; ++Col)
{
        for(Row= 0; Row<= Max_Row; ++Row)
        {
                X = Y = xn = yn = 0;
                Count =1;
                While(Count<Max_Iterations) & (X+Y < 4)
                {
                        X = xn2 ;
                        Y = yn2 ;
                        yn = 2xnyn  + Q[Row];
                        xn = X – Y + p ;
                        Count = Count +1;
                }

Tô màu điểm ảnh (Col, Row) bởi màu có số hiệu
                    Count mod   Max_Colors ;
        }
p = p + Δp;

}
```

Hình 11.2 thể hiện tập Mandelbrot cổ điển với các giá trị khảo sát nằm trong vùng giới hạn bởi X_{min} = -2.0, Y_{min} = -1.2, X_{max} = 1.2, Y_{max} = 1.2 và Max_Iterations = 512, Max_Colors = 1.6.



Hình 11.2

II.8 TẬP JULIA:

□ **Đặt vấn đề:**

Đối với biểu thức $z_{n+1} = z_n^2 + c$, ngoài hướng đã khảo sát như đã trình bày trong phần tập Mandelbrot, còn có hướng khảo sát khác bằng cách cho c cố định và xem xét dãy (z_n) ứng với mỗi giá trị khác của z_0. Theo hướng này chúng ta sẽ thu được 1 lớp các đối tượng fractal mới được gọi là tập Julia.

Tập Julia và tập Mandelbrot là hai lớp các đối tượng fractal có mối liên hệ rất chặt chẽ với nhau. Một tính chất đáng chú ý là tập Mandelbrot có thể xem như một loại "bản đồ" Mandelbrot có thể cho ra các dạng tập Julia đầy sức lôi cuốn. Các vị trí như vậy được quan sát thấy ở gần biên của tập Mandelbrot. Nhất là gần các chỏm nhọn. Ngoài ra khi phóng to một phần của tập Mandelbrot, ta sẽ thu được một hình rất giống với tập Julia được tạo bởi giá trị của tâm phần được phóng to.

□ **Công thức toán học:**

Để thể hiện tập Julia trên màn hình máy tính, ta vẫn sử dụng các công thức như trong phần tập Mandelbrot, như là:

$$x_{n+1} = x_n^2 - y_n^2 + p$$
$$y_{n+1} = 2x_n y_n + q$$

Ngoài ra các tính chất đã nêu về giới hạn của dãy (z_0) vẫn được sử dụng cho tập Julia.

□ **Thuật toán thể hiện tập Julia:**

Điểm khác biệt so với tập Mandelbrot ở đây là giá trị p và q được giữ cố định, mặt phẳng màn hình biến đổi thành mặt phẳng phức thu hẹp biểu diễn các giá trị của x_0 với:

- Trục x biểu diễn phần thực của số phức z_0.
- Trục y biểu diễn phần ảo của số phức z_0.

Ngoài ra còn có sự phân lớp các giá trị của z_0 như sau:

Lớp 1:

Bao gồm các giá trị (z_0) có $| z_k | < 2$, với $0 \le k \le N$ trong đó N là hằng số hữu hạn. Tức là lớp 1 gồm các giá trị z_0 làm cho dãy (z_0) không tiến ra vô cực.

Lớp 2:

Bao gồm các giá trị (z_0) có $| z_n | > 2$, với $n \ge k$, $k \in Z^+$, tức là gồm các giá trị làm cho dãy (z_n) tiến ra vô cực.

Ngược lại với tập Mandelbrot, khi thể hiện tập Julia trên màn hình, chúng ta quan tâm đến các giá trị z_0 làm cho dãy (z_n) không hội tụ đến vô cực. Do đó kỹ thuật tô màu của tập Julia vẫn là kỹ thuật xoay vòng nhưng hoàn toàn ngược lại với kỹ thuật tô màu tập Mandelbrot. Trong kỹ thuật tô màu này:

- Các điểm ảnh tương ứng với các giá trị z_0 thuộc lớp 1, sẽ được gán màu tùy thuộc độ lớn của $| z_l|$ với l là ngưỡng quyết định hội tụ của dãy (z_n) đã nêu trong định nghĩa về lớp 1.
- Các điểm ảnh tương ứng với giá trị z_0 thuộc lớp 2 sẽ được gán màu trùng với màu nền của bảng màu đang sử dụng.

Với các thay đổi như vậy, tập Julia sẽ được thể hiện bằng thuật toán trình bày như sau:

Thuật toán tổng quát để thể hiện tập Julia:

Gồm các bước sau:

Bước 1:

Xuất phát với 1 giá trị khởi đầu $z_0 = (x_0, y_0)$ và giá trị cố định c = (p, q).

Bước 2:

Kiểm tra z_0 thuộc lớp 1 hay 2.

Bước 3:

Tô màu điểm ảnh tương ứng với z_0 theo kỹ thuật tô màu được nêu ở trên.

Bước 4:

Chọn giá trị z_0 mới và lặp lại bước 1 cho đến khi đã quét hết tất cả các giá trị z_0 cần khảo sát.

Sử dụng ký hiệu đã được xác định khi trình bày thuật toán Mandelbrot chúng ta có thuật toán tạo tập Julia một cách chi tiết được viết dưới dạng sau:

```
for(Col = 0; Col<Max_Col; ++Col)
{
for(Row=0; Row<= Max_Row; ++Row)
        {
                x0 = xmin + Col* Δx0;
                y0 = ymax - Row* Δy0;
                X=Y= 0;
                Count =1;
                While((Count < Max_Iterations) & (X+Y < 4))
                {
                        X = xn^2 ;
                        Y = yn^2 ;
                        y0 = 2x0y0  + q ;
                        x0 = X – Y + p ;
                        ++Count ;
                }
                if(Count > Max_Iterations)
                        Tô màu điểm ảnh (Col, Row) bởi màu có số hiệu
                        (X + Y) (Max_Color – 1) mod (Max_Color +1) ;
                else
                        Tô màu điểm ảnh (Col, Row) bởi màu nền của bảng
                        màu hiện tại;
        }
```

Sự khác biệt chủ yếu giữa thuật toán này với thuật toán Mandelbrot là sự thay đổi vai trò của z_0 và c. Giá trị c = (p,q) được giữ cố định trong khi z_0 thay đổi. Các đại lượng Δx_0, Δy_0, x_0, y_0 được xác định theo cách hoàn toàn giống với các đại lượng Δp, Δq, p, q trong thuật toán tạo tập Mandelbrot tức là:

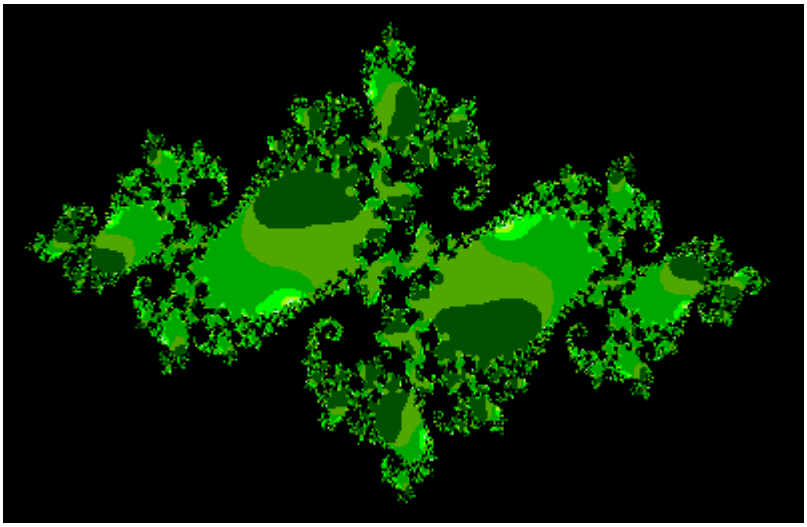
$$\Delta x = \frac{x_{\max} - x_{\min}}{Max\_Col}$$

$$\Delta y = \frac{y_{\max} - y_{\min}}{Max\_Row}$$

$$x_0 = x_{min} + Col * \Delta x_0;$$
$$y_0 = y_{max} - Row * \Delta y_0;$$

Còn có một điểm cần chú ý là các giá trị x_0, y_0 vừa đại diện cho z_0 ban đầu và cũng đại diện cho dãy (z_n) trong vòng lặp kiểm tra z_0 thuộc lớp 1 hay 2.

Hình minh hoạ tập Julia như sau:



II.9 HỌ CÁC ĐƯỜNG CONG PHOENIX

Họ các đường cong Phoenix do Shigehiro Ushiki ở trường đại học Kyoto tìm ra. Phương trình của đường cong được xác định bởi:

$$Z_{n+1} = z_n^2 + p + q.z_{n-1}$$

Trong đó:

$$Z_i \in C \quad \forall i, N.$$

$$p = (p, 0) \in C.$$

$$q = (q, 0) \in C.$$

Phương trình được khai triển thành các phần thực và ảo của z_n có dạng:

$$x_{n+1} = x_n^2 - y_n^2 + p + q.x_{n-1}$$

$$y_{n+1} = 2x_n.y_n + q.y_{n-1}$$

với: $x_{n+1} = Re(z_{n+1});$

$$y_{n+1} = Im(z_{n+1}).$$

Khi đó việc thể hiện đường cong này lên màn hình gần giống với việc thể hiện tập Julia. Tuy nhiên có hai điểm thay đổi quan trọng:

Thay đổi 1:

- Trục x của màn hình biểu thị phần ảo của số phức z_0.
- Trục y của màn hình biểu thị phần thực của số phức z_0.

Ở đây chúng ta đảo ngược các trục thực và ảo của mặt phẳng phức thông thường là để thể hiện hình ảnh theo chiều đứng chứ không phải chiều ngang. Hình 14.1 trình bày 1 loại đường cong loại này với yêu cầu rõ ràng là phải đổi vai trò của trục x và y để hình ảnh có thể được thể hiện tốt trên màn hình. Do đó tương ứng với một điểm ảnh (Col, Row) trên màn hình sẽ là số phức z = (x, y) có dạng:

$$x = y_{max} - Row * \Delta x;$$

$$y = y_{min} - Col * \Delta y;$$

Với:

$$\Delta x = \frac{y_{\max} - y_{\min}}{Max\_Row}$$

$$\Delta y = \frac{x_{\max} - x_{\min}}{Max\_Col}$$

Thay đổi 2:

Thay đổi về thuật toán tô màu. Ở đây với các điểm thuộc lớp 1 (theo định nghĩa đã nêu ở phần về tập Julia) chúng ta sẽ sử dụng 3 loại màu tuỳ theo ngưỡng hội tụ:

Màu 1: được sử dụng để tô các điểm z_0 cho ra giá trị $| z_k | < 2$ với tối đa k = 32 lần lặp.

Màu 2: được sử dụng để tô các điểm z_0 cho ra giá trị $| z_k | < 2$ với số lần lặp từ 33 đến 64.

Màu 3: được sử dụng để tô các điểm z_0 cho ra giá trị $| z_k | < 2$ với số lần lặp vượt quá 64 lần.

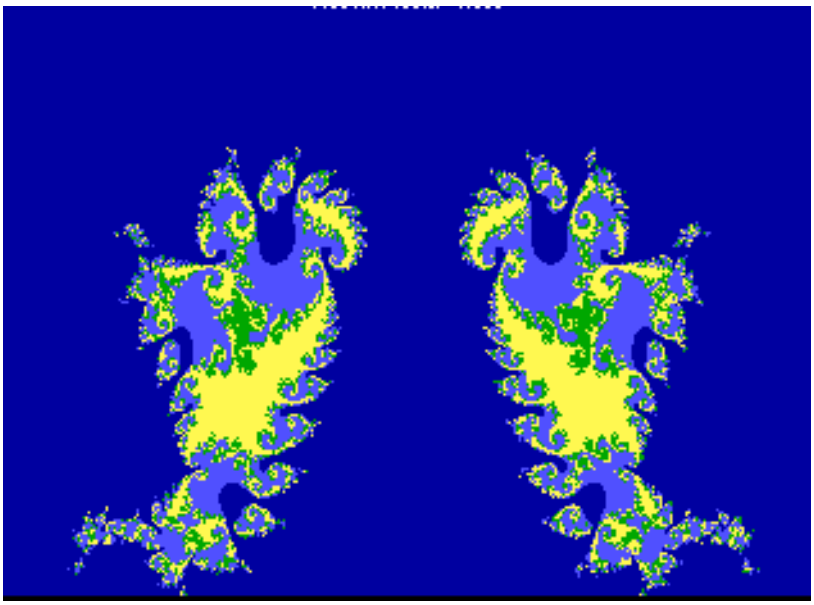
Còn đối với các điểm thuộc lớp 2, chúng ta sẽ tô chúng bằng một màu khác với màu nền hiện tại.

Với các thay đổi như vậy, đoạn mã dùng xác định giá trị z_0 thuộc lớp 1 hay 2, cùng với kỹ thuật tô màu điểm ảnh sẽ được viết dưới dạng:

```
for(Col = 0; Col<Max_Col; ++Col)
{
        for(Row=0; Row<= Max_Row; ++Row)
        {
                xn = ymax - Row* Δx;
                yn = xmin + Col* Δy;
                X =Y= 0;
                Count = 0;
```

```
While((Count < Max_Iterations) & (X+Y < 4))
{
        X = x_n^2 ;
        Y = y_n^2 ;
        y_{n+1} = 2x_n y_n  + q y_{n-1};
        y_{n-1} = y_n ;
        x_{n+1} = X– Y + qx_{n-1} + p;
        x_{n-1} = x_n;
        x_n = x_{n+1};
        y_n = y_{n+1};
        ++Count;
}
if(Count > Max_Iterations)
Tô màu điểm ảnh (Col, Row) bằng màu dành cho các điểm loại 2;
        else
                if (Count >= 64)
                Tô màu điểm (Col, Row) bằng màu 3;
                else
                        if (Color >= 32)
                        Tô điểm ảnh (Col, Row) bằng màu 2;
                        else
                        Tô điểm ảnh (Col, Row) bằng màu 1;
    }
}
```

Đây là ảnh của đường cong Phoenix



Hình 14.1: Đường cong Phoenix

Nhận xét

Tam giác Sierpinski , Tấm thảm Sierpinski , Hình cầu Sierpinski, Tứ diện Sierpinski , Bọt biển Menger là những hình "có xương mà không có thịt". Sau vô hạn lần áp dụng quy tắc sinh , những hình trên có diện tích tiến về 0.

Đặc trưng phổ biến của các hình Fractal

Tự đồng dạng

Qua các hình đã xem xét như đường con von Koch và đường cong Minkowski , Tam giác Sierpinski, Bọt biển Menger , ... những hình mà Maldelbrot gọi là "hình Fractal" , chúng ta thấy đó là những hình rất khác lạ. Chúng rất khác với những hình cổ điển trong hình học Euclid (tam giác , đa giác , hình tròn , ...) Những hình Fractal có cấu trúc nhìn bề ngoài rất phức tạp ,dù xét ở bộ phận nhỏ đến đâu cũng có những chi tiết hết sức tinh vi. Thế nhưng những hình phức tạp này lại hình thành bằng cách thực hiện lặp đi lặp lại "quy tắc sinh" khá đơn giản.

Đặc trưng phổ biến của nhiều hình Fractal là có thể phân tích ra thành những bộ phận nhỏ tuỳ ý, mà mỗi bộ phận này là một bản sao thu nhỏ của toàn thể Fractal. Mandelbrot gọi đó là tính *tự đồng dạng*.

Thứ nguyên phân số

Trong hình học Euclide ta coi điểm là đối tượng không có "kích thước", không có chiều dài, chiều rộng lẫn chiều cao, điểm có số chiều là 0 (hay thứ nguyên 0). Đường thẳng chỉ có chiều dài , không có chiều rộng, không có chiều cao, đường thẳng có số chiều là 1 (thứ nguyên 1). Mặt phẳng có số chiều là 2 (thứ nguyên 2), vì chỉ có chiều dài và chiều rộng, không có chiều cao. Không gian có số chiều là 3 (thứ nguyên 3)

Thứ nguyên của một đối tượng hình học có thể hiểu theo một cách khác

Xét các hình: đoạn thẳng, hình vuông, hình lập phương. Có thể chia mỗi hình này ra N phần để cho mỗi phần đồng dạng với hình ban đầu. Gọi tỷ số đồng dạng là 1/κ thì ta có N=k đối với đoạn thẳng, $N=k^2$ đối với hình vuông và $N=k^3$ đối với hình lập phương.

Như vậy tổng quát $N=k^d$, d là số thứ nguyên

Từ $N=k^d$ => d= logN/logk

Ví dụ: Đường cong Minkowski. Tập sinh của nó sau lần đầu tiên áp dụng quy tắc sinh gồm có 8 đoạn thẳng (N=8), mỗi đoạn thẳng đồng dạng với hình ban đầu theo tỷ số ¼ (k=4), do đó thứ nguyên của Đường cong Minkowski là

log8/log4=1,5

Tam giác Sierpinski có thứ nguyên d=log3/log2=1,585... vì tập sinh gồm 3 phần (N=3) , mỗi phần đồng dạng với tam giác ban đầu theo tỷ số ½ (k=2)

KẾT LUẬN CHƯƠNG

Hình học Fractal mới xuất hiện khoảng 1 thập kỷ nhưng đã nhanh chóng trở thành một công cụ nghiên cứu lý thú và có hiệu lực mạnh mẽ ở hầu hết các lĩnh vực khoa học , kỹ thuật , nghệ thuật: từ toán học, vật lý, thiên văn, sinh lý học , tâm lý học, ngôn ngữ học cho đến công nghệ thông tin và truyền thông, kinh tế tài chính, kiến trúc, xây dựng, ngay cả đến âm nhạc, hội hoạ , văn học cũng tìm thấy ở Fractal nhiều điểm tương đồng.

Hiện nay trên thế giới Fractal rất phát triển. Thị trường tranh fractal và phần mềm sáng tác fractal cũng không hề nhỏ, nó có giá trị gần 1 tỉ USD trong năm 2004. Tìm hiểu sâu hơn về fractal hãy truy cập những địa chỉ sau: www.les.stclair.btinternet.co.uk

www.biofractalevolution.com

www.fractalism.com

.

Còn muốn tạo ra những hình họa fractal nhanh chóng và dễ dàng nhất, hãy sử dụng các phần mềm miễn phí:

- Aros Fractals: Dung lượng chỉ 165 KB, tương thích với mọi môi trường Windows hiện hành, giải nén vào một thư mục rồi sử dụng ngay mà không cần cài đặt. Tải về từ địa chỉ www.arosmagic.com.

- Double Fractal: Của tác giả Joao Paulo Schwarz Schuler. Dung lượng 676 KB, không cần cài đặt, tải về từ địa chỉ www.schulers.com/fractal.

TÀI LIỆU THAM KHẢO

[1] Michale Barnsly. " ***Fractal Everywhere*** ", University of Wisconsin-Madison, 1998.

[2] John C.Hart "***Fractal Image Compression and the Inverse Problem of Recurrent Iterated Function Systems***". School of EECS, 1995

[3] Ngô Quốc Tạo, "***Các vấn đề cơ bản của hình học Fractal và ứng dụng trong nén ảnh***", .đề tài CS97-12, 1997.

[4] http://search.dmoz.org/cgi-bin/search?search=fractal

[5] http://chaos4.phy.ohiou.edu/~thomas/frac_frame.html.