2. TEORI PENUNJANG

2.1. *2D Computer Graphics*

2D computer graphics adalah gambar digital yang dihasilkan oleh komputer, khususnya yang berbentuk dua dimensi karena hanya mempunyai sumbu koordinat X dan Y saja. Gambaran *2D computer graphics* kira-kira adalah seperti sebuah kanvas kosong (pada komputer dapat dianggap sebagai *frame buffer* yang berupa kumpulan *array of pixels)* berwarna putih (sebagai latar belakang warna, pada komputer latar belakang biasanya berwarna hitam). Kanvas itu selanjutnya dapat digambari dan diwarna sesuai keinginan (pada komputer penggambaran dan pewarnaan sama dengan mengisi *array of pixels* dengan bilangan tertentu sebagai penunjuk warna).

2D computer graphics dapat berupa *vector graphics, raster graphics* ataupun kombinasi diantara keduanya. *Raster graphics* adalah penyimpanan gambar 2D pada komputer dilakukan secara apa adanya dalam bentuk *array of pixels* dan sering kali disebut sebagai *bitmap. Sedangkan vector graphics* adalah proses penyimpanan gambar 2D pada komputer dilakukan secara vektor, maksudnya penyimpanan berupa posisi titik-titik yang kemudian saling dihubungkan dengan garis, ataupun lingkaran sehingga membentuk gambar.

2.1.1. *Image Processing*

Image processing adalah suatu metode yang digunakan untuk memproses atau memanipulasi *image* dalam bentuk dua dimensi. Segala operasi untuk memperbaiki, menganalisa, atau mengubah suatu gambar disebut *image processing.* Konsep dasar pemrosesan suatu objek yang menggunakan *image processing* diambil dari kemampuan indera penglihatan manusia yang selanjutnya dihubungkan dengan kemampuan otak manusia.

Dalam sejarahnya, *image processing* telah diaplikasikan dalam berbagai bentuk, dengan tingkat kesuksesan yang cukup besar. Seperti berbagai cabang ilmu lainnya, *image processing* menyangkut pula berbagai gabungan cabang-

cabang ilmu, diantaranya adalah optik, elektronik, matematika, fotografi, dan teknologi komputer.

Beberapa faktor menyebabkan perkembangan sistem *image processing* menjadi berkembang pesat pada saat ini. Salah satu yang utama adalah dibutuhkannya suatu teknologi yang dapat bekerja secara mandiri, dalam arti teknologi yang dapat memproses data-data yang diterima dan pada akhirnya teknologi tersebut harus bisa mengambil keputusan sendiri dari hasil pengolahan data sebelumnya. Selain itu penurunan biaya akan perlatan komputer yang dibutuhkan serta peningkatan tersedianya peralatan untuk proses tampilan gambar juga menjadi salah satu faktor semakin berkembangnya *image processing*.

Pada umumnya, objektifitas dari *image processing* adalah mentransformasikan atau menganalisis suatu gambar sehingga informasi baru tentang gambar dibuat lebih jelas.

Ada empat klasifikasi dasar dalam *image processing* yaitu *point, area, geometric,* dan *frame*.

a. *Point* memproses nilai *pixel image* berdasarkan nilai atau posisi dari *pixel* tersebut. Contoh dari proses *point* adalah *adding, subtracting, contrast stretching,* dan lainnya.
b. *Area* memproses nilai *pixel-pixel* suatu *image* berdasarkan nilai *pixel* tersebut beserta nilai *pixel* di sekelilingnya. Contoh dari proses *area* adalah *convolution, blurring, sharpening,* dan *filtering.*
c. *Geometric* digunakan untuk merubah posisi dari *pixel-pixel.* Contoh dari proses *geometric* adalah *scaling, rotation, mirroring.*
d. Sedangkan *frame* memproses nilai *pixel* berdasarkan operasi dari dua buah *image* atau lebih. Contoh dari proses *frame* adalah *addition, subtraction,* dan *and/or*.

2.1.2. Sistem Warna RGB

Cahaya adalah suatu bentu energi elektromagnetik yang terdiri dari spektrum frekwensi yang mempunyai panjang gelombang kurang lebih dari 400 nanometer untuk cahaya *ultraviolet* sampai kurang lebih 700 nanometer untuk

cahaya merah. Warna dari suatu objek merupakan pantulan cahaya yang tidak dapat diserap oleh objek tersebut.

Teori tentang cahaya ini dapat diperoleh dari teori tentang optik dari Isaac Newton, yang menyatakan bahwa suatu warna dapat terpancar karena perbedaan panjang gelombang sehingga menghasilkan perbedaan efek visual. Newton menyatakan bahwa terdapat tujuh warna utama (merah, jingga, kuning, hijau, biru, nila, dan ungu) dan dipakai untuk menghitung warna dari spektrum panjang gelombang yang lainnya.

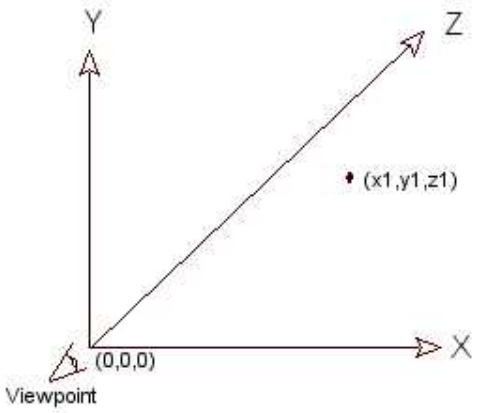
Semua warna yang ada merupakan perpaduan dari 3 (tiga) macam warna primer yaitu : Warna merah (*Red*), warna hijau (*Green)*, Warna biru (*Blue).* Perpaduan dari ketiga warna primer ini dipakai pada sistem warna RGB. Bila ketiga warna primter dicampur, maka akan dihasilkan suatu warna tertentu, tergantung dari komposisi ketiga warna primer tersebut.

Gambar pada sistem *digital* dapat diwakili dengan format RGB untuk setiap titiknya, di mana setiap komponen warna diwakili dengan satu *byte.* Jadi untuk masing-masing komponen R, G dan B mempunyai variasi dari 0 sampai 255. Total variasi yang dapat dihasilkan untuk sistem warna *digital* ini adalah 256x256x256 atau 16.777.216 jenis warna. Karena setiap komponen warna diwakili dengan satu *byte atau* delapan *bit,* maka total *bit* yang digunakan untuk merepresentasikan warna RGB adalah 8+8+8 atau 24 *bit.*

2.2. *3D Computer Graphics*

3D computer graphics adalah pengembangan dari *2D computer graphics. 3D computer graphics* mempunyai tiga sumbu koordinat yaitu X, Y, Z yang datanya disimpan pada komputer namun untuk representasi hasil dari *3D computer graphics* harus dilakukan kalkulasi untuk mengubah tiga sumbu koordinat menjadi dua sumbu koordinat.

Seringkali pada dunia *computer graphics software*, perbedaan antara keduanya menjadi tidak jelas. Beberapa aplikasi 2D menggunakan teknik 3D untuk memperoleh efek seperti *lightning,* sebaliknya beberapa aplikasi 3D menggunakan tampilan teknik 2D.



Gambar 2.1. Sumbu koordinat 3D

2.2.1. Proyeksi 3D ke 2D

Pada dasarnya komputer hanya dapat menampilkan gambar dalam bentuk dua dimensi. Oleh karena itu diperlukan adanya proyeksi untuk mengubah kumpulan *array* titik yang mempunyai 3 sumbu menjadi 2 sumbu koordinat agar dapat ditampilkan di layar monitor. Setelah proyeksi selesai, kumpulan *array* yang sekarang hanya mempunyai 2 sumbu, dapat ditampilkan dengan *pixel plotting* di layar monitor.

x 2D = HW + 3D x * VD / 3D z;
y 2D = HH + 3D y * VD / 3D z; (2.1)

Keterangan :

- HW = setengah dari lebar resolusi layar monitor
- HH = setengah dari tinggi resolusi layar monitor
- VD = perkiraan jarak pandang mata

2.2.2. Transformasi

Benda 3D seharusnya bisa untuk dipindah, diputar, dilihat dari berbagai macam sudut pandang, oleh karena itu diperlukan transformasi, yaitu untuk mengubah atau memindah posisi titik pada dunia 3D. Posisi titik pada dunia 3D

dapat dianggap sebagai vektor (x,y,z), sehingga untuk mengubah posisi dari (x,y,z) itu diperlukan suatu penambahan, pengurangan ataupun perkalian matriks. Transformasi dibagi menjadi 3 macam yaitu berdasarkan operasi yang berkenaan dengan matriks :

- Translasi

  Translasi berarti memindah posisi titik dengan penambahan jarak. Transformasi ini melibatkan penjumlahan vektor

- Skala / *Scaling*

  Scaling berarti menskala ulang baik menskala menjadi lebih besar, ataupun menskala lebih kecil. *Scaling* melibatkan operasi perkalian vektor.

- Rotasi

  Rotasi berarti merotasi / memutar posisi titik ke posisi titik baru dengan suatu sumbu tertentu sebagai porosnya. Formula rotasi pasti melibatkan operasi sin dan cos yang berulang-ulang, hal ini sangat memperlambat proses, karena perhitungan sin dan cos yang dilakukan komputer sangat lambat. Salah satu trik yang sering digunakan adalah menggunakan *array* untuk menyimpan sin dan cos dari 0 sampai 360.

2.2.3. *Polygon*

Polygon berasal dari kata *poly* yang berarti banyak dan kata *gonos* yang berarti sudut. *Polygon* adalah suatu kurva tertutup yang terdiri dari tiga garis lurus atau lebih. Garis lurus yang membentuk *polygon* dinamakan sisi, dan pertemuan antara sisi *polygon* dinamakan puncak.

Polygon dinamai sesuai dengan jumlah sisi ditambah dengan kata ‘gon’ dibelakangnya, contohnya *pentagon, dodecagon,* kecuali segitiga. Matematikawan menuliskan jumlah sisi ditambah dengan kata ‘gon’ dibelakangnya, contohnya : 17-gon. Kadang-kadang variabel pun dapat digunakan untuk menyebutkan *polygon*, contohnya n-gon.

Pada dunia *2D computer graphics, polygon* dapat digambar dengan hubungan garis-garis lurus yang direpresentasikan lewat *pixel-pixel* yang saling berurutan sehingga nampak seperti garis lurus. Pertemuan antara sisi *polygon* (puncak) biasanya diwakili dengan sebuah *pixel.*

Pada dunia *3d computer graphics*, *polygon* yang paling sering digunakan adalah segitiga, karena segitiga selalu mendifinisikan satu sisi datar saja. Segitiga yang dibuat berdasarkan pada 3 titik 3D yang mempunyai sumbu x,y,z, kemudian diproyeksikan menjadi dua dimensi. Kita akan memperoleh 3 titik 2D yang siap untuk dihubungkan dengan garis lurus membentuk segitiga di layar monitor.

Gambar 2.2. Gabungan *polygon* segitiga yang membentuk suatu objek

2.2.4. *Lightning* dan *Shading*

Dunia *3D computer graphics* terdiri dari banyak sekali titik-titik, kemudian titik-titik itu dihubungkan menjadi *polygon* yang berbentuk segitiga-segitiga. Seperti yang disebutkan di atas untuk menampilkan segitiga 3D itu, perlu dilakukan proyeksi menjadi 2D agar dapat tampil dilayar monitor. Kemudian segitiga-segitiga yang dihasilkan pada layar 2D itu diwarna sehingga segitiga-segitiga itu saling bersatu menjadi suatu kesatuan objek 3D yang utuh.

Agar warna yang ditampilkan oleh segitiga itu tambah lebih realistik, kita harus membedakan warna segitiga yang satu dengan yang lain berdasarkan pencahayaannya itulah yang disebut sebagai *lightning*, contohnya segitiga yang menghadap cahaya pasti lebih terang warnanya daripada yang tidak mendapat cahaya. Untuk mendapatkan intensitas warna pada suatu segitiga, digunakan vektor normal. Untuk memperoleh vektor normal, digunakan rumus *cross product* antar vektor.

$A \times B = C$

$Cx=Ay*Bz - By*Az$

$Cy=Az*Bx - Bz*Ax$

$Cz=Ax*By - Bx*Ay$ (2.5)

Caranya adalah dengan mengambil puncak segitiga berdasarkan arah jarum jam, kemudian mengurangi yang tengah dengan kedua puncak lainnya, dengan itu kita mendapatkan dua vektor baru yang dapat di *cross product*. Vektor baru yang dihasilkan harus dijadikan hanya mempunyai panjang 1. Karena itu digunakan rumus

$l=sqrt(x^2+y^2+z^2)$

$normal=(x/l,y/l,z/l)$ (2.6)

Proses ini lambat, karena melibatkan akar kuarat dan pembagian. Vektor normal cukup dihitung satu kali ketika kita menginsialisasi suatu objek 3D, dan merotasikan bersama-sama dengan tiap titik segitiga. Selanjutnya kita harus mengetahui sudut antara vektor normal dengan vektor cahaya, caranya adalah dengan operasi *dot product.*

$A \bullet B = (Ax*Bx + Ay*By + Az*Bz)$

$cos(S)=(N \bullet L) / (|N| * |L|)$ (2.7)

Keterangan :

$|N|$ adalah panjang vektor N (normal)

Hasil dari cos(S) adalah nilai antara -1 sampai 1, nilai ini dapat digunakan untuk menentukan intensitas untuk *lightning.* Kita juga dapat menambahkan

ambient, yaitu warna minimal untuk suatu objek, agar objek tersebut tidak sepenuhnya hitam bila nilai cos(S) adalah 0. Caranya adalah dengan :

Intensitas = *Ambient* + maksimum_warna * cos(S) (2.8)

Setelah semua penjelasan di atas, kita bisa mendapatkan segitiga-segitiga berintensitas warna yang membentuk suatu objek dengan nama *flat shading* objek.

Gambar 2.3. Bola tanpa *shading*

Gambar 2.4. Bola dengan *flat shading*

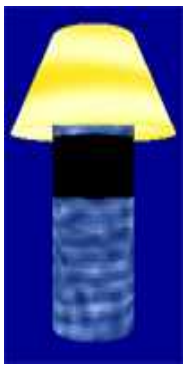
2.2.5. *Texture Mapping*

Pada penjelasan di atas, setiap polygon yang berbentuk segitiga akan diwarna sesuai intensitas warna tertentu, namun kekurangannya adalah tiap segitiga hanya akan dapat menampilkan satu warna saja yang diperoleh berdasarkan perhitungan intensitas garis normal warna.

Texture Mapping adalah metode *shading* yang berbeda dengan *flat shading*. Secara kasarnya metode *texture mapping* adalah menampilkan gambar pada *polygon*. Jadi satu *polygon* yang sebelumnya hanya mempunyai satu warna berdasarkan intensitas warna kini dapat mempunyai banyak warna membentuk suatu gambar tertentu tetapi warna-warna itu tetap memiliki intensitas berdasarkan perhitungan garis normal.

Dengan adanya *texture mapping* suatu objek 3D dapat ditampilkan dengan lebih realistis, misalnya untuk menampilkan dadu, akan lebih masuk akal jika pada tiap sisinya bergambar angka 1-6 daripada dadu yang hanya berupa kubus dengan warna yang sama tanpa pengenal angka 1-6. Contoh lain adalah lampu tanpa tekstur dengan lampu yang bertekstur.

Gambar 2.5. Lampu tanpa tekstur



Gambar 2.6. Lampu dengan tekstur

Tekstur pada *texture mapping* berupa gambar dua dimensi yang biasanya disimpan pada suatu file. Jadi gambar dua dimensi itu dapat dikatakan ditempelkan pada *polygon-polygon* yang tersebar sehingga membentuk *polygon-polygon* dengan banyak warna. Gambar dua dimensi dengan ukurannya apapun dapat dijadikan sebuah teksture, karena gambar dua dimensi yang ditempelkan pada *polygon* dapat menyesuaikan, apakah harus diperbesar ataupun diperkecil skalanya.

2.3. *OpenGL API*

OpenGL (Open Graphics Library) adalah adalah suatu API (*Application Programming Interface)* untuk berbagai macam bahasa pemrograman yang dapat digunakan untuk membuat aplikasi *3D Computer Graphics* maupun *2D Computer Graphics.*. *Interface OpenGL* mencakup 250 fungsi yang dapat digunakan untuk membuat aplikasi *3D Computer Graphics* mulai dari yang sederhana hingga yang amat kompleks.

OpenGL adalah suatu *interface program* untuk mengakses fungsi-fungsi yang disediakan oleh *driver* suatu *hardware graphics card* untuk komputer. Jadi *OpenGL* memanfaatkan *graphics card* untuk melakukan perhitungan seperti *rendering, shading, lightning, rotation, translation,* yang bila tanpa adanya *OpenGL,* fungsi-fungsi harus dibuat secara manual dengan rumus-rumus matematika yang sudah dijelaskan di atas. Dengan perhitungan yang dilakukan

secara *hardware* oleh *graphics card*, hasilnya adalah sesuatu yang lebih cepat dan lebih bagus untuk digunakan dalam pembuatan aplikasi 3D.

OpenGL sebenarnya merupakan pengembangan dari *IRIS GL interface.* Kode *IRIS GL* hanya mendukung akses pada fungsi yang ada pada *hardware* saja, sehingga bila suatu aplikasi membutuhkan misalnya *texture mapping*, tetapi *hardware* tidak mendukungnya, maka *IRIS GL* tidak dapat menjalankan aplikasi itu. *OpenGL* merupakan pengembangan lebih lanjut dari *IRIS GL* agar dapat mengatasi masalah seperti itu dengan menyediakan fungsi misalnya *texture mapping* secara *software* sehingga aplikasi tetap dapat berjalan.

OpenGL dibuat untuk dapat mendukung banyak bahasa pemrograman, diantaranya adalah *Java, Fortran, Pike, Perl, Ada, Phyton, Delphi, Visual Basic, Visual C++*, dan masih banyak lagi. *OpenGL* juga didukung oleh berbagai macam sistem operasi seperti *Windows, UNIX* dan *MacOS.*

OpenGL didesain khusus untuk menjadi program yang hanya dapat menampilkan output, yaitu menyediakan fungsi-fungsi *rendering*. Inti *interface OpenGL* tidak menyediakan konsep sistem *windows, audio, printing, keyboard mouse*, atau *input devices* yang lain. Kode program *OpenGL* dapat dikatakan berdiri sendiri tanpa campur tangan sistem operasi.

2.4. Komponen *GLScene*

GLScene adalah komponen untuk *delphi* yang merupakan sebuah komponen untuk menghasilkan aplikasi berbasis *OpenGL. GLScene* menyederhanakan keseluruhan fungsi-fungsi yang ada pada *OpenGL*, sehingga dengan menggunakan *GLScene* ini, kita dapat membangun aplikasi *3D computer graphics* dengan sangat mudah, karena kita tinggal menempelkan komponen, lalu memanfaatkan fungsi-fungsinya tanpa harus mengetahui fungsi-fungsi dasar *OpenGL*.

Contohnya adalah untuk animasi, dengan *GLScene* ini animasi cukup dilakukan dengan *meload* suatu format file tertentu seperti **.smd*, lalu *GLScene* secara otomatis membaca file tersebut dan menterjemahkannya menjadi animasi 3D, tanpa kita harus tau cara pembacaan filenya. Dengan *GLScene* ini kita cukup

mengisi properti variabel untuk tiap animasi, maka animasi dapat berjalan sendiri, bahkan secara *real-time* dengan program yang kita tulis lainnya.

GLScene juga menyamarkan inisialisasi yang sebelumnya harus diperlukan bila membuat aplikasi 3D dengan *OpenGL*. Contohnya adalah inisialisasi untuk membuat *window*. Jadi dengan *GLScene* kita tidak perlu repot-repot membuat sebuah *window / form* yang dapat digambari, cukup dengan menempelkan komponen tanpa inisialisasi apapun kita sudah dapat membuat aplikasi 3D yang siap untuk di*compile* dan dijalankan.

Berikut ini adalah unit-unit yang tambahan yang disediakan oleh *GLScene* :

- *GLCamera*

  GLCamera digunakan untuk menunjukkan posisi kamera yang melihat ke arah objek 3D. *GLCamera* ini harus selalu ada pada komponen *GLScene* yang baru ditempelkan, karena merupakan kamera yang menunjuk ke sesuatu. Tidak perlu adanya inisialisasi *property* awal untuk *GLCamera* ini, karena semua nilai sudah diisi secara *default*, tetapi *GLCamera* tetap dapat dipindah secara manual melalui perintah-perintah :

  ```
  Glcamera.position.x:=20;
  Glcamera.positon.y:=20;
  Glcamera.position.z:=20;
  ```

- *GLLightSource*

  GLLightSource digunakan untuk memberikan pencahayaan atau penyinaran pada suatu benda. Model dari pencahayaan dapat diatur melalui *property LightStyle.* Sama seperti *GLCamera, GLLightSource* juga tidak memerlukan inisialisasi awal karena semuanya sudah terisi secara *default.*

- *GLDummyCube*

  GLDummyCube digunakan hanya sebagai *dummy* untuk menandakan bahwa sesuatu yang berada dibawah *GLDummyCube* ini adalah suatu kesatuan yang utuh. Contohnya adalah bila dibawah *GLDummyCube* terdapat objek 3D berbentuk kelinci dan macan, lalu diluar *GLDummyCube* ada objek 3D berbentuk anjing, lalu kita merotasi *GLDummyCube* maka yang ikut terotasi hanyalah objek kelinci dan macan saja, anjing tidak terotasi karena anjing

berada diluar *GLDummyCube*. Sama seperti yang lain, *GLDummyCube* tidak memerlukan inisialisasi awal, karena semua nilai sudah terisi secara *default.*

- *GLHUDSprite*

  GLHUDSprite digunakan untuk menampilkan gambar dua dimensi pada dunia tiga dimensi dengan syarat gambar dua dimensi itu benar-benar sebuah gambar dua dimensi yang tidak ikut dalam perhitungan rotasi, translasi, dan sebagainya, sehingga *GLHUDSprite* dapat digunakan untuk menunjukkan sesuatu tanpa harus berbentuk 3D, contohnya sebagai latar belakang yang tidak ikut bergerak, atau sebagai informasi jumlah nyawa yang tersisa pada sebuah game. *GLHUDSprite* memerlukan inisialisasi untuk mengambil gambar dari sebuah file, menetapkan posisi, dan menetapkan lebar, tinggi gambar.

  Contoh perintah :

  ```
  hudsprite1.material.texture.image.loadfromfile('tekstur.bmp');
  hudsprite1.material.texture.disabled:=false;
  hudsprite1.setsize(640,480);
  hudsprite1.position.x:=320;
  hudsprite1.position.y:=240;
  hudsprite1.visible:=true;
  ```

  Perintah di atas adalah untuk mengambil gambar dari file ‘tekstur.bmp’ kemudian menampilkannya dilayar dalam bentuk dua dimensi dengan lebar 640 dan tinggi 480 *pixel,* dan meletakkannya tepat di posisi x = 320, dan y = 240.

- *GLMesh*

  GLMesh digunakan untuk meletakkan titik pada dunia 3D dan menampilkannya pada dunia 3D dengan menggabungkan titik-titik itu menjadi *polygon. GLMesh* dapat digunakan untuk menampilkan *polygon* dalam bentuk segitiga maupun segiempat.

  Contoh perintah :

  ```
  function setpoint(const x, y,z  : Single) : TAffineVector;
  begin
   SetVector(Result, x, y, z);
  ```

```
end;
p1::=setpoint(1, 1, 1);
p2:=setpoint( 2, 2, 2);
p3:=setpoint(( 3, 3, 3);
mesh1.AddVertex(p1, NullVector, 2);
mesh1.AddVertex(p2, NullVector, 2);
mesh1.AddVertex(p3, NullVector, 2);
```

Perintah di atas adalah untuk menampilkan *polygon* bentuk segitiga dengan koordinat (1,1,1) , (2,2,2) dan (3,3,3) dan dengan warna 2.

2.5. *Fractal*

Fractal dalam arti tertentu dapat disebut sebagai *fragment*, tetapi arti sebenarnya adalah suatu objek atau kuantitas yang menampilkan kesamaan diri sendiri (*self-similarity)*. Ada juga yang menyebutkan *fractal* adalah pola pengulangan yang menampilkan kekompleksan lebih dalam jika diperbesar. Sedangkan menurut penulis, *fractal* adalah suatu proses pengulangan yang rekursif dimana input yang dipakai adalah hasil output dari sebelumnya.

Sebelum adanya penemuan komputer, *fractal* sempat menjadi pertanyaan penting sebanyak dua kali bagi dunia. Yang pertama adalah ketika orang Inggris membuat peta untuk mengukur panjang pantai Inggris. Dari jauh, pengukuran garis pantai sekitar 5000. Untuk pengukuran jarak menengah menghasilkan perkiraan sekitar 8000. Dan dengan pengukuran yang sangat detail dan lebih dalam pada peta panjang garis pantai bisa mencapai dua kali panjang sebenarnya. Peta Inggris pada peta dunia tidak mencatat secara keseluruhan pantai dan pelabuhan yang ada, sedangkan pada peta Inggris saja pencatatan tentu saja dilakukan dengan lebih lengkap mencakup hampir keseluruhan pantai dan pelabuhan. Semakin dekat lagi kita lihat, semakin detail dan panjanglah garis pantai itu. Orang Inggris itu hanya tahu sedikit bahwa ini adalah bagian dari *fractal*.

Yang kedua adalah ketika *pre-computer fractals* diketahui oleh matematikawan Perancis bernama Gaston Julia. Dia bertanya-tanya bagaimana

bentuk fungsi kompleks polinomial seperti z^2+c, dimana c adalah konstanta bilangan kompleks yang terdiri dari bilangan riil dan bilangan imaginer.

Kemudian *Benoit Mandelbrot*, seorang karyawan IBM, berpikir untuk menulis sebuah program dengan formula z^2+c dan kemudian menjalankannya pada sebuah komputer IBM. *Mandelbrot* mendapatkan sebuah gambar yang sangat indah. *Mandelbrot* adalah orang pertama yang berhasil menggunakan komputer untuk melakukan pekerjaan repetisi dan pengulangan penghitungan *fractal* dan merepresentasikan dalam bentuk pixel.

Secara keseluruhan *fractal* bisa dibagi menjadi 2 macam, keduanya menampilkan kesamaan yaitu *self-similarity* pada tiap gambarnya :

a. *Fractal* yang menggunakan rumus matematika untuk menghasilkan gambar yaitu seperti contoh di atas *Mandelbrot* dengan rumus z=z^2+c, dimana c adalah bilangan kompleks.
b. *Fractal* yang tidak menggunakan rumus matematika. Pada *fractal* ini tidak digunakan bilangan kompleks, hanya digunakan perulangan secara rekursif dan biasanya menggunakan bilangan acak sebagai pembantu. Contoh dari *fractal* ini adalah *Koch Curve, Sierpinsky Gasket.*

Berikut ini adalah penjelasan mengenai bilangan kompleks, *fractal* dengan atau tanpa rumus matematika secara lebih mendalam :

2.5.1. Bilangan Kompleks

Adalah suatu bilangan yang mempunyai bentuk a + bi, dimana A dan B adalah bilangan riil, dan i adalah bilangan imaginer (akar kuadrat dari -1). Bilangan kompleks dapat dianggap sebagai bilangan yang mempunyai dua dimensi, karena mempunyai bilangan real dan bilangan imaginer. Kadang-kadang bilangan kompleks ditulis (a,b) sama seperti penulisan koordiant *Cartesian*.

Penjumlahan pada bilanga kompleks :

$(a+bi) + (c+di) = (a+c) + (b+d)i$ atau

$(a,b) + (c,d) = (a+b,c+d)$ (2.9)

Perkalian pada bilangan kompleks :

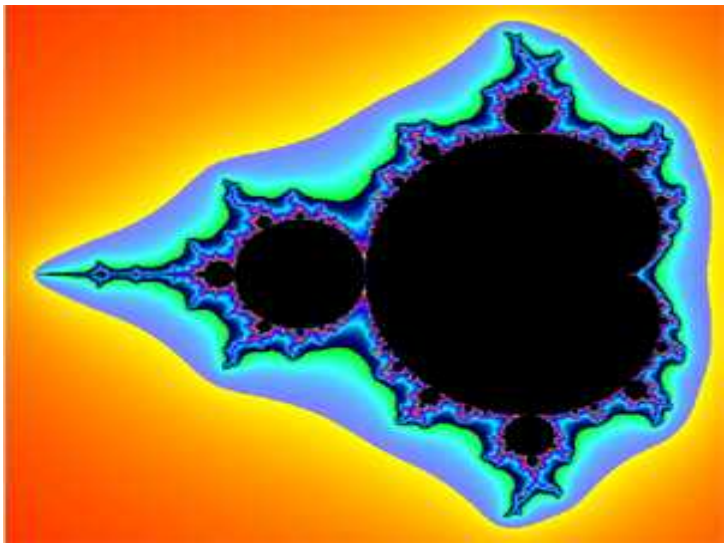
(a+bi)*(c+di) = (ac-bd) + (ad+bc)i atau

(a,b) * (c,d) = (ac-bd,ad+bc) (2.10)

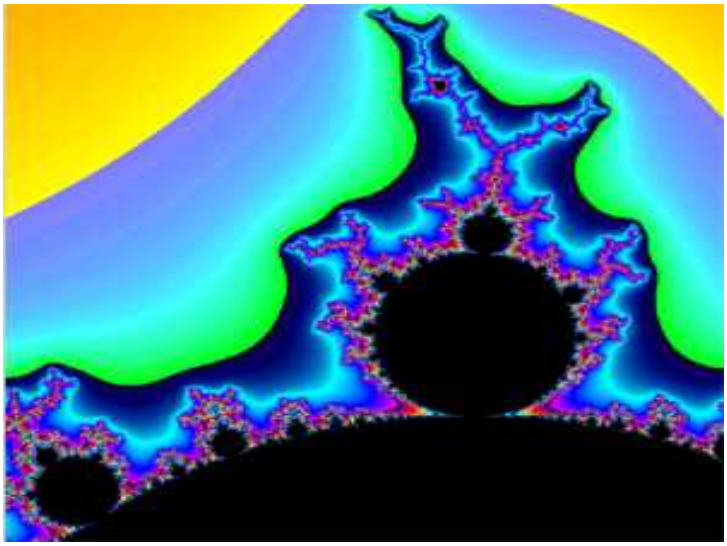
2.5.2. *Fractal* dengan Rumus Matematika, *Mandelbrot Set*

Mandelbrot set adalah suatu penempatan titik C pada perulangan rumus Zn+1 = Zn*Zn +C, dimana C adalah bilangan kompleks. *Mandelbrot* termasuk salah satu objek terkompleks yang pernah ditemukan pada matematika, tetapi sebaliknya *Mandelbrot* adalah sebuah formula yang sederhana saja. Mengali Z dengan diri sendiri, kemudian menambahkannya dengan C, dan hasilnya disimpan pada Z yang baru (perkalian Z dan penambahan C dilakukan dengan cara perkalian dan penambahan pada bilangan kompleks).

Mandelbrot set terdiri dari bilangan kompleks. Setiap bilangan pada bilangan kompleks a+bi menyatakan : a adalah jarak ke kiri atau ke kanan dari titik tengah (negatif untuk kiri, positif untuk kanan), dan b adalah jarak ke atas atau ke bawah dari titik tengah (negatif untuk bawah, positif untuk atas), dan i menyatakan bilangan imaginer akar kuarat -1.



Gambar 2.7. *Mandelbrot set* sebelum *zoom in*

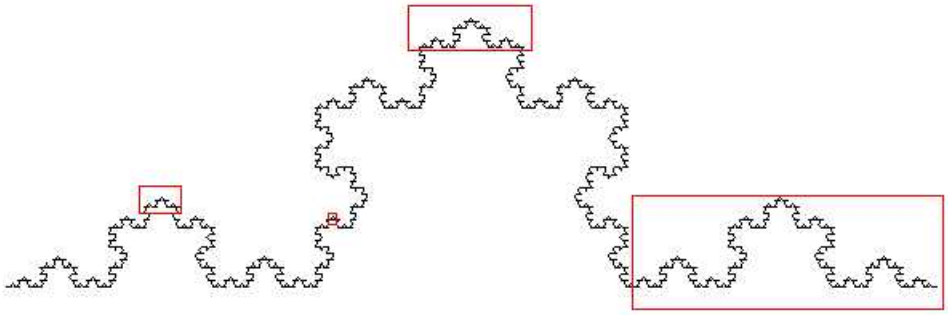


Gambar 2.8. *Mandelbrot set* setelah di *zoom in 4x* pada bagian atas

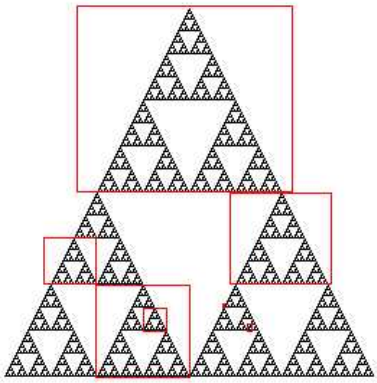
Mandelbrot mempunyai *self-similarity* yaitu pada bagian-bagiannya bila dilakukan *zoom in* secara terus menerus kita akan menjumpai gambar-gambar yang memiliki kesamaan dengan gambar awal.

2.5.3. *Fractal* tanpa Rumus Matematika

Seperti yang disebutkan di atas, *fractal* juga bisa dihasilkan tanpa rumus matematika. *Fractal* tanpa rumus matematika mengandalkan perulangan dan operasi rekursif. Pada *fractal* jenis ini, *self similarity* akan tampak lebih jelas dibandingkan dengan *fractal* dengan rumus matematika. Contoh *fractal* ini adalah *Koch Curve, Sierpinski Gasket.*



Gambar 2.9. *Koch Curve*

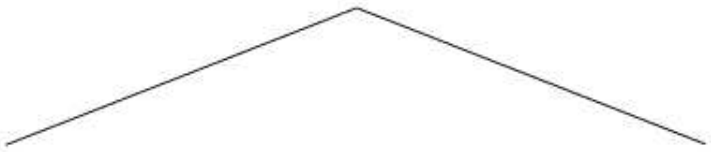


Gambar 2.10. *Sierpinski Gasket*

2.6. *Midpoint Displacement Algorithm*

Midpoint Displacement termasuk dalam model *fractal* tanpa rumus matematika. Algoritma intinya adalah untuk memperoleh bilangan random dengan batasan tertentu setiap iterasi, dan pada setiap iterasi berikutnya batasan random itu dikurangi menjadi lebih kecil dengan dikali koefisien *roughness* (H). Rumus H sebenarnya adalah : 2^-H. Jadi untuk H = 1, perhitungan menjadi 2^-1 = 1/2, jadi setiap iterasi berikutnya batasan random dikali ½ lebih kecil.

Untuk lebih jelasnya, pada iterasi pertama kita ambil contoh suatu bilangan random Yrandom dengan batasan sebesar -1 sampai 1. Hasil random tiap iterasi disimpan pada suatu *array* yaitu *array* Y yang menunjukkan ketinggian kurva. Misal setelah random, Yrandom menunjukkan hasil -1, maka -1 dianggap menjadi titik dengan ketinggian Y=-1. Hasilnya adalah sebagai berikut.



Gambar 2.11. *Midpoint Displacement* pada iterasi pertama

Pada iterasi kedua, kita ingin membagi lagi dua garis yang dihasilkan pada iterasi pertama dengan cara mengambil titik tengah dari dua garis itu (jadi ada 2

titik tengah), kemudian menambahkannya dengan Yrandom yang baru (dihasilkan dengan batasan baru). Batasan baru maksudnya adalah batasan yang telah dikali dengan H=-1 (jadi 2^-1 = ½), jadi batasan yang dimaksud sudah berkurang pada iterasi kedua menjadi -½ sampai ½.

Gambar 2.12. *Midpoint Displacement* pada iterasi kedua

Untuk iterasi-iterasi selanjutnya, hal yang dilakukan adalah sama, yaitu membagi setiap garis yang ada dengan mengambil titik tengahnya, kemudian menambahkannya dengan Yrandom baru yang sudah mempunyai batasan baru setelah dikali dengan H. Pada iterasi ketiga contohnya, garis yang harus dibagi ada sebanyak 4 garis, setelah iterasi ketiga garis menjadi 8 buah.

Gambar 2.13. *Midpoint Displacement* pada iterasi ketiga

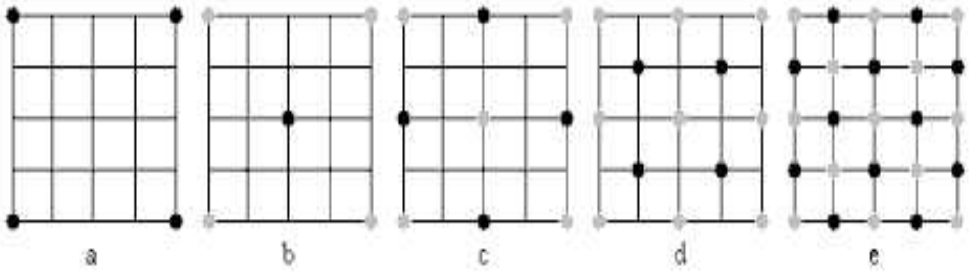
Pada pemrograman umumnya, seperti juga penulis, koefisien *roughness* (H) diganti menjadi rumus pembagian biasa tanpa adanya 2^-H. Alasannya adalah sebagai penyesuaian dengan program nantinya, karena 2^-H mempunyai arti 1/(2^H) sehingga pecahan hanya dapat berupa kelipatan 2^n, sedangkan pada proses pembuatan permukaan batu, setiap bilangan desimal memberikan hasil yang berbeda walaupun bilangan desimal itu hanya berbeda sangat sedikit saja.

2.7. *Diamond Square Algorithm*

Diamond Square Algorithm juga termasuk pada bagian *fractal* tanpa rumus matematika. *Diamond Square Algorithm* merupakan pengembangan dari *Midpoint Displacement Algorithm*. *Midpoint Displacement* berdimensi satu, karena hasil *Midpoint Displacement* dapat disimpan pada *array* satu dimensi (hasil tampilan berupa gambar kurva dua dimensi. Sedangkan *Diamond Square Algorithm* berdimensi dua karena harus disimpan pada *array* dua dimensi (hasil tampilan berupa gambar kurva tiga dimensi).

Algoritma ini dimulai dengan membuat *array* kosong dua dimensi sebesar NxN, dengan N = 2^X+1. X dapat berupa sembarang nilai yang tidak terlalu besar, karena untuk X=9 saja *array* menjadi 513x513, sangat besar dan boros memory. Kemudian algoritma akan melakukan perhitungan pada *array* NxN itu dengan cara *square* / kotak segiempat ataupun *diamond* / wajik sehingga disebut *diamond square algorithm.*

Untuk contoh, kita mengambil *array* 5x5 (X=2), kemudian dilakukan langkah2 pengisian *array* pada posisi seperti dibawah ini :



Gambar 2.14. Pengisian *array* pada *Diamond Square Algorithm*

Pada gambar a, kita mencari hasil random pada 4 titik yaitu pojok-pojok berupa segiempat. Seperti pada *Midpoint Displacement* kita mencari nilai random dari batasan -1 sampai 1.

Kemudian kita mencari posisi tengah dari 4 titik tadi lewat perpotongan garis diagonal antara 4 titik tadi (gambar b). Posisi tengah itu diisi dengan rata-rata nilai dari 4 titik gambar a ditambah dengan bilangan random yang sudah

dikali batasannya dengan H sehingga menjadi lebih kecil (sama seperti *Midpoint Displacement).*

Pada gambar c, kita mencari lagi titik baru yaitu dengan cara mengambil titik dari gambar b menariknya ke atas, bawah, kiri dan kanan, sehingga kita mendapatkan 4 titik baru. Keempat titik itu dihitung dengan cara menghitung rata-rata nilai titik yang ada pada gambar b dan a (membentuk wajik dengan titik tengah berupa titik baru yang akan dicari itu) ditambah dengan nilai random dari batasan yang sudah dikali H. Sehingga untuk menghitung gambar c, diperlukan 4 kali perhitungan berbentuk wajik.

Pada gambar d, kita melakukan rekursif untuk menghitung 4 titik tengah dari 9 titik yang sudah dihasilkan pada gambar c. Titik tengah dihitung dengan cara yang sama persis seperti pada gambar b (dengan cara rekursif), tetapi pada gambar d ini, kita melakukan 4 kali perhitungan titik tengah, karena pada gambar c sudah tercipta 4 segiempat. Hal ini berlaku juga pada gambar e, yang perhitungannya sama seperti gambar c.

Perhitungan secara rekursif ini dilakukan terus menerus sampai seluruh *array* dua dimensi terisi penuh. Dari hasil *array* dua dimensi inilah penulis akan mencoba untuk menciptakan tiruan gambar dengan pola batu-batuan. Agar lebih realistis, penulis menggabungkannya dengan *3D computer graphics*, sehingga pola batu dapat tampak jauh ataupun dekat tergantung pada posisi x,y,z yang dihasilkannya.

2.8. *Borland Delphi*

Delphi adalah suatu aplikasi dan bahasa pemrograman berbasis bahasa Pascal yang dikembangkan oleh Borland. Delphi memiliki keunggulan–keunggulan dibanding bahasa pemrograman lainnya, diantaranya adalah :

- Menggunakan bahasa pemrograman yang bersifat *high-level language.* Terstruktur dengan jelas dan mudah dimengerti.
- Merupakan aplikasi permrograman yang bersifat *Rapid Aplication Deployment* (RAD) yang berarti Delphi difokuskan untuk bisa membuat aplikasi-aplikasi komputer dengan cepat dengan menggunakan *Object Oriented Programming*.

- Penggunaan yang *user friendly*, dan fasilitas dokumentasi yang tersedia dengan lengkap dan terstruktur dengan rapi.
- Banyaknya *component-component* yang tersedia pada Delphi maupun yang didapat dari *internet* mampu memperluas kemampuan Delphi.