PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK PEMBANGKIT ERRAIN MAP DENGAN METODE FRAKTAL BERBASIS OpenGL

Disusun Oleh :

ANAS MUZAKIR

NRP. 2693 100 010

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK PEMBANGKIT RRAIN MAP DENGAN METODE FRAKTAL BERBASIS OpenGL

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
S u r a b a y a

Mengetahui / Menyetujui

osen Pembimbing I

Dosen Pembimbing II

27/02'99

Kupersembahkan untuk :

ibunda dan *ayahanda* tercinta
yang selalu memberikan cintanya

ABSTRAK

Grafika komputer saat ini telah menjadi salah satu element dalam visualisasi bagai macam obyek. Banyak algoritma yang dikembangkan untuk menghasilkan mbar yang efektif, cepat, dan sesuai dengan realita. Salah satu dari algoritma ebut adalah metode fraktal yang digunakan untuk menghasilkan gambar yang ndekati kemiripan dengan obyek permukaan alam. Dari segi perangkat lunak telah yak dikembangkan paket grafik, salah satunya adalah OpenGL yang embangkan oleh Silicon Graphics Inc. OpenGL yang telah menjadi salah satu ndar dalam pemrosesan grafika komputer.

Metode fraktal menggunakan sifat *self-similarity*, dimana tiap-tiap bagian dari yek fraktal mempunyai kesamaan bentuk dengan skala yang berbeda. Hal ini, yang njadikan fraktal banyak digunakan untuk merepresentasikan permukaan alam.

Pembuatan *terrain map* dilakukan dengan mendefinisikan nilai ketinggian uk tiap titik. Setelah data ketinggian didapat maka gambar dari terrain tersebut bat ditampilkan baik dalam bentuk gambar rangka (*wire frame*), dan gambar yang warna dengan proses *rendering*. Proses *rendering* yang dilakukan dimulai dengan mberikan warna untuk ketinggian tertentu dengan metode *Gouraud shading*, elah proses pewarnaan dilakukan maka didapatkan tekstur yang dapat diterapkan uk berbagai macam data ketinggian yang berbeda.

Gambar yang dihasilkan oleh perangkat lunak menunjukkan bentuk rmukaan yang mendekati kenyataan pada alam.

KATA PENGANTAR

Segala puji syukur penulis panjatkan kepada Allah SWT., Tuhan semesta
n, karena tanpa kehendak-Nya, segala usaha untuk menyelesaikan Tugas Akhir ini
k akan pernah terwujud. Ucapan terima kasih juga kami sampaikan kepada semua
ak yang telah membantu terselesaikannya Tugas Akhir ini.

Tugas Akhir ini dibuat untuk memenuhi persyaratan mendapatkan gelar
jana Teknik Informatika pada jurusan Teknik Informatika, Institut Teknologi
uluh Nopember ini, berjudul :

"PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK

PEMBANGKIT TERRAIN MAP

DENGAN METODE FRAKTAL BERBASISKAN OpenGL"

Tugas Akhir ini dibuat untuk memberikan masukan pada bidang grafika
mputer, terutama pada bidang simulasi dan animasi. Penulis berharap Tugas Akhir
memberikan manfaat kepada pembaca, khususnya kepada mereka yang tertarik
da grafika komputer dan penerapannya.

Pada kesempatan ini penulis mengucapakan terima kasih atas bantuannya

Rekan-rekan senasib dan sepenanggungan C09, pengurus Lab KOMISI, "kakak-kakak" dan "adik-adik", terima kasih atas perhatiannya.

Rekan-rekan TA-wan dan TA-wati.

Komunitas "The Dji Sam Soe Apartment", dengan canda yang diberikan.

Pihak-pihak yang penulis tidak dapat sebutkan satu-persatu.

Penulis menyadari bahwa tugas akhir ini bukanlah karya yang terbaik, karena uk mencapai hal tersebut amatlah sulit, karena keterbatasan penulis sebagai insan g mempunyai berbagai macam kekurangan, tetapi penulis telah berusaha wujudkanya dengan segala daya upaya yang terbaik.

Surabaya, Pebruari 1999

Penulis

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

DAFTAR LAMPIRAN

Halaman

BAB I

BAB I

PENDAHULUAN

1.1. LATAR BELAKANG

Komputer telah menjadi suatu alat yang berkembang cepat untuk menghasilkan gambar, sehingga tak mengherankan jika grafika komputer telah berkembang sangat luas, baik penggunaan pada industri, bisnis, pemerintahan, seni, maupun pada dunia pendidikan.

Seiring berkembangnya grafika komputer, yang mampu menampilkan gambar berbagai macam obyek, maka bermunculan macam-macam metode matematis yang diterapkan untuk mempresentasikan obyek tersebut kedalam layar komputer.

Metode fraktal telah banyak digunakan dalam grafika komputer untuk pembangkitan kurva dan permukaan. Teknik ini banyak digunakan untuk menampilkan gambar agar tampak lebih alamiah, seperti api, gunung, pohon, rumput dan lain-lain. Sifat kemiripan dari dirinya sendiri (*self-similarity)* yang terdapat pada obyek-obyek yang dihasilkan dengan metode fraktal merupakan

Self-similarity dari suatu obyek fraktal mempunyai suatu bagian yang merupakan pengskalaan dari seluruh bagian dari obyek tersebut. Dimulai dari bentuk awal, dibangun bagian-bagian obyek lainnya dengan memberikan faktor skala tertentu terhadap seluruh bagian obyek tersebut. Dapat digunakan faktor skala yang sama untuk keseluruhan proses penyekalaan ataupun faktor skala yang berbeda untuk tiap-tiap bagian.

Suatu obyek fraktal mempunyai dua karakteristik yang mendasar yaitu tingkat kerincian yang tak terbatas pada setiap bagian dan self-similarity antara bagian-bagian pada obyek. Sifat self-similarity dari suatu obyek dapat dilihat dari dari berbagai bentuk, tergantung dari pilihan dari representasi fraktal. Suatu obyak fraktal dijabarkan dengan suatu prosedur yang menspesifikasikan operasi yang diulang-ulang untuk menghasilkan rincian dari bagian obyek.

Terrain map adalah suatu fungsi sederhana yang mengambarkan sebuah topografi dari suatu daerah. Fungsi yang memproyeksikan sebuah koordinat 2 dimensi (x,y), kesebuah variabel ketinggian dan variabel warna.

Pembentukan obyek tiga dimensi sebagai representasi dari obyek yang sebenarnya pada dunia nyata ke dalam layar komputer, membutuhkan banyak perhatian disamping menentukan koordinat dunia nyata ke layar. Hal lain yang

Suatu hasil gambar pada grafika komputer yang mendekati bentuk aslinya akan memuat dua hal penting yaitu : representasi obyek grafika yang akurat dan pendeskripsian dari proses pewarnaan yang mendekati kenyataan.

OpenGL merupakan kumpulan fungsi-fungsi untuk grafika tiga dimensi dan grafika *rendering*, yang tidak terpengaruh oleh peralatan dan sistem operasi tertentu. OpenGL dikembangkan oleh Silicon Graphics Inc. (SGI) yang penggunaannya diperuntukkan pada workstation mereka. Semenjak itu OpenGL telah berkembang sangat cepat dan diimplementasikan pada berbagai macam sistem operasi dan perangkat keras, dalam hal ini termasuk sistem operasi Windows.

1.2. TUJUAN PENELITIAN

Tujuan dari tugas akhir ini adalah merancang dan membuat sebuah perangkat lunak yang mampu menghasilkan obyek 3 dimensi berupa permukaan (terrain map), sedangkan metode yang digunakan untuk membangun terrain map adalah metode fraktal, penggunaan OpenGL sebagai basis dari pemrograman, dengan kata lain merupakan dasar pembentukan gambar. Adapun gambar hasil yang dibentuk dapat berupa gambar rangka (*wire frame*), sampai pada

1. Membangun suatu obyek terrain map baik yang berbentuk *wire frame* maupun suatu obyek tiga dimensi yang terlah mengalami proses *rendering*.
2. Menggunakan metode fraktal sebagai metode yang dipakai untuk menghasilkan terrain map, yang tidak terlepas dari penerapan unsur matematis.
3. Implementasi perangkat lunak yang meliputi desain struktur data dan implentasinya terhadap bahasa pemrograman dan juga menerapkan *OpenGL* sebagai basis pembentukan perangkat lunak.

Pembatasan permasalahan, diberlakukan pada pemilihan algoritma metode fraktal yang dipilih guna menghasilkan sebuah terrain map, yaitu menggunakan metode *Random Midpoint Displacement* atau dikenal juga sebagai metode *Diamond Square Algorithm*.

Perangkat lunak yang dikembangkan berfungsi sebagai pembangkit terrain map, dengan kemampuan menampilkan gambar yang berbentuk gambar rangka *(wire frame)* dan gambar yang telah mengalami proses *rendering*. Proses *rendering* yang dilakukan terbatas pada pewarnaan *(shading)* dan penempelan pola *(texturing)*. Proses *shading* akan menghasilkan warna yang sesuai dengan

1.4. METODOLOGI PENELITIAN

Untuk menyelesaikan permasalahan ini, proses penyelesaian dibagi sebagai berikut :

- **Studi literatur**

  Pada tahap ini dilakukan pengumpulan data dan studi literatur, baik literatur yang mengenai pembentukan terrain map dan teori fraktal, juga literatur mengenai pembuatan perangkat lunak.

- **Perencanaan program**

  Pada tahap ini dilakukan perencanaan struktur data, algoritma dan diagram alur untuk program yang dibuat. Perancangan perangkat lunak yang dibuat berbasiskan pada pemrograman berorientasi obyek.

- **Pembuatan program**

  Pada tahap ini dilakukan implementasi program yang telah direncanakan pada tahap sebelumnya.

- **Evaluasi dan revisi program**

  Pada tahap ini dilakukan evaluasi dan perbaikan dari program yang telah dibuat.

- **Penulisan naskah**

1.5. SISTEMATIKA PEMBAHASAN

Pembahasan laporan Tugas Akhir ini disusun dengan sistematika sebagai berikut :

Bab I, Pendahuluan. Menjelaskan latarbelakang, tujuan, pembatasan masalah dan metodologi pembuatan perangkat lunak.

Bab II, Teori Penunjang. Membahas secara garis besar prosedur pembentukan obyek tiga dimensi ke dalam 2 dimensi, dan juga dibahas tentang proses pewarnaan pada sebauh obyek, semua prosedur yang dilakukan pada OpenGL.

Bab III, Fraktal dan Terrain Map. Membahas tentang sistem fraktal, pembahasan dimulai dari definisi, sampai komponen-komponen yang terdapat dalam fraktal, pembentukan *terrain map* dengan menggunakan metode fraktal.

Bab IV, Perancangan dan pembuatan perangkat lunak. Menguraikan rancangan sistem perangkat lunak yang dibuat berdasarkan pendekatan-pendekatan yang telah diuraikan dalam bab II dan bab III

Bab V, Uji Coba dan Evaluasi. Melaporkan hasil uji coba sistem perangkat lunak yang telah jadi dan mengevaluasi keandalan sistem. Dalam hal ini gambar

BAB I

TEORI PENUNJANG

BAB II

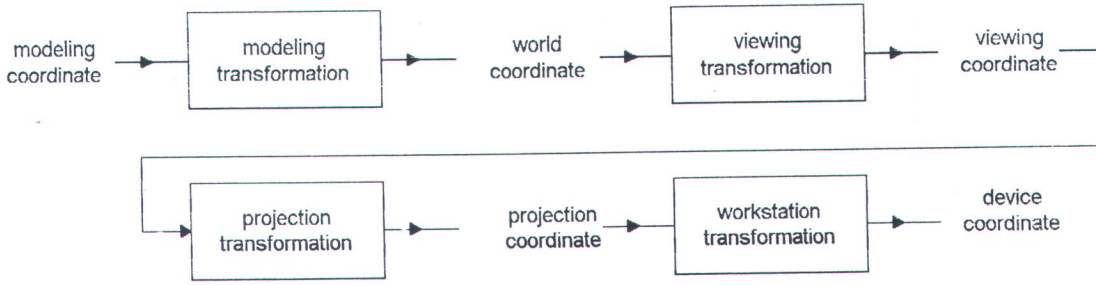
TEORI PENUNJANG

Dalam bab ini dibahas mengenai OpenGL, bagaimana menampilkan bidang tiga dimensi ke dalam bidang dua dimensi. Perintah apa saja yang terkait pada OpenGL untuk melaksanakan proses transformasi. Dan juga dibahas mengenai proses pewarnaan, bagaimana sebuah permukaan yang dikenai cahaya memantulkan kembali cahaya tersebut. Selain itu juga dibahas pemberian pola tekstur pada sebuah bidang

2.1. Menampilkan Obyek 3 Dimensi

Langkah untuk membuat suatu tampilan dari gambar tiga dimensi ke dalam bidang dua dimensi seperti layar komputer, dapat dianalogikan dengan proses pengambilan gambar oleh sebuah kamera. Untuk mendapatkan sebuah gambar, langkah pertama adalah menempatkan kamera pada posisi yang diinginkan, selanjutnya mengorientasikan kamera ke arah obyek yang akan diambil gambarnya, proses ini dinamakan *viewing transformation*. Setelah penempatan kamera selesai, proses selanjutnya adalah menempatkan obyek yang

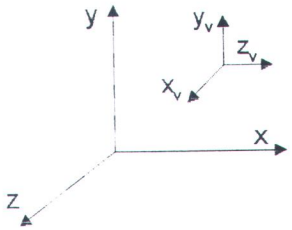
Proses-proses di atas dapat dilihat pada gambar 2.1. Masukan untuk tiap proses adalah keluaran dari proses sebelumnya, hanya saja pada saat pertama proses, masukan adalah nilai koordinat model pada sistem koordinat dunia.



Gambar 2.1 Transformasi tiga dimensi ke bidang dua dimensi, dari koordinat model sampai ke koordinat layar[1].

2.1.1. Transformasi Pandangan (*Viewing Transformation*)

Transformasi ini memindahkan semua titik suatu obyek, dari koordinat dunia ke koordinat pandang *(viewing coordinate)*. Proses yang dilakukan adalah dengan menempatkan sebuah bidang pandang (analogi dari sebuah kamera) pada suatu titik ruang tiga dimensi dan mengorientasikan kamera tersebut ke obyek yang akan difoto. Untuk melakukan viewing transformation dibutuhkan sebuah sistem koordinat pandang *(viewing coordinate system)*, seperti yang dapat dilihat



Gambar 2.2 Sistem koordinat pandang dengan sumbu x_v,y_v dan z_v yang relatif terhadap koordinat dunia

Sebuah bidang pandang *(view plane)* ditempatkan pada sistem koordinat pandang. Pada bidang pandang ini ditentukan sebuah titik pada koordinat dunia, sebagai titik pusat dari sistem koordinat pandang, dan dikenal sebagai titik acuan pandang (*view reference point,* VRP). Arah normal dari bidang pandang disebut sebagai normal bidang pandang (*view plane normal*, VPN), yang dinyatakan sebagai vektor satuan **N**.

Selanjutnya ditentukan arah atas untuk sistem tersebut dengan menempatkan suatu vektor satuan **V**, vektor ini dikenal sebagai *view up vector.* Fungsi vektor ini untuk menentukan arah positif sumbu y_v. Dengan menggunakan kedua vektor **V** dan **N** dapat ditentukan sebuah vektor satuan **U** untuk menentukan arah positif x_v.

Sebelum obyek dapat diproyeksikan ke bidang pandang, perlu ditentukan

2. Melakukan rotasi terhadap sistem koordinat pandang, sehingga berhimpit dengan sistem koordinat dunia.

Jika VRP ditempatkan pada (x, y, z) pada sistem koordinat dunia, maka matriks translasi titik tersebut adalah :

$$T = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Cara mendapatkan matriks rotasi adalah dengan menghitung unit vektor *u*, *v*, dan *n*, berdasarkan vektor **N** dan **V**.

$$n = \frac{N}{|N|} = (n1, n2, n3) \qquad (2\text{-}1)$$

$$u = \frac{V \times N}{|V \times N|} = (u1, u2, u3) \qquad (2\text{-}2)$$

$$v = n \times u = (v1, v2, v3) \qquad (2\text{-}3)$$

Dari persamaan di atas didapatkan sebuah matriks rotasi

$$R = \begin{bmatrix} u1 & u2 & u3 & 0 \\ v1 & v2 & v3 & 0 \\ n1 & n2 & n3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Menggunakan perintah yang ada pada modeling transformation yaitu `glTranslate()` dan `glRotate()`. Kedua perintah tersebut dijelaskan pada bagian transformasi model.
- Menggunakan perintah `gluLookAt()`. Perintah ini melakukan proses translasi dan rotasi.

```
void gluLookAt(Gldouble eyex, Gldouble eyey, Gldouble eyez,
Gldouble centerx, Gldouble centery, Gldouble centerz, Gldouble
upx, Gldouble upy, Gldouble upz,)
```

Parameter *eyex, eyey, eyez,* menunjukan titik VRP. Sedangkan *centerx, centery, centerz* menunjukan sebuah titik tengah dari gambar, dimana mata melihat kepada titik tersebut. Dan parameter *upx, upy, upz,* menunjukan arah ke atas guna mencari vektor **U.**

Sebelum menjalankan perintah-perintah di atas maka perintah `glMatrixMode(GL_MODEVIEW)` perlu dipanggil terlebih dahulu. Dengan demikian OpenGL akan melakukan perhitungan transformasi pandangan dari masukan kedua cara tersebut.

2.1.2. Transformasi model (*Modeling Transformation*)

Pada transformasi ini, dilakukan proses penempatan dan pembentukan

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

parameter tx, ty dan tz, masing-masing menunjukan jarak translasi untuk dimensi koordinat x, y, dan z.

Perintah untuk translasi pada OpenGL dilakukan dengan `glTranslate*()`, dengan ketentuan

```
void glTranslate{fd}(TYPEx, TYPEy, TYPEz)
```

Dimana TYPEx, TYPEy, dan TYPEz menentukan besaran jarak translasi untuk masing-masing arah x, y, z.

- Rotasi

Operasi matriks untuk rotasi pada sumbu z dengan sudut putar θ adalah :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

untuk rotasi pada sumbu x :

$$\begin{bmatrix} x' \\ \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

OpenGL menyediakan perintah rotasi ini dengan perintah

```
glRotate*()
```

```
void glRotate{fd}(TYPE angle, TYPEx, TYPEy, TYPEz);
```

perintah ini akan melukan operasi rotasi dengan besaran putaran sebesar *angle*, dengan arah rotasi ditentukan oleh x, y, z.

- Skala

Operasi matriks untuk proses penyekalaan dapat dijabarkan sebagai berikut :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

dengan Sx, Sy, Sz adalah parameter skala.

Perintah OpenGL adalah `void glScale{fd}(TYPESx, TYPESy, TYPESz)`

2.1.3 Transformasi Proyeksi (*Projection Transformation*)

garis lurus. Untuk proyeksi persfektif, semua posisi obyek ditransformasikan ke bidang pandang sepanjang garis yang mengarah pada satu titik yang disebut pusat proyeksi.

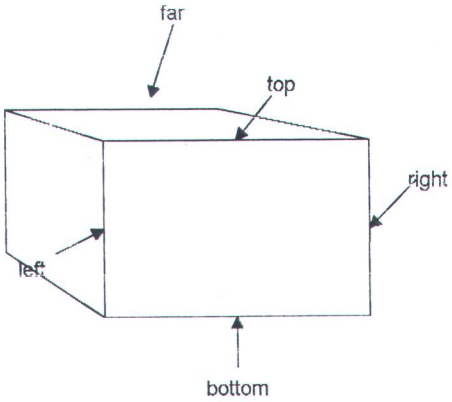
Pada OpenGL sebelum mendefinsikan model proyeksi, maka perlu memanggil perintah `glMatrixMode`(GL_PROJECTION). Dengan demikian OpenGL akan melakukan perhitungan pembuatan matriks transformasi proyeksi. Agar matriks yang dihasilkan tidak terpengaruh oleh perhitungan matriks proses sebelumnya maka `glLoadIdentity`() dijalankan terlebih dahulu, sesudah perintah `glMatrixMode`().

2.1.3.1. Proyeksi Orthographic (*Orthographic Projection*)

Persamaan transformasi untuk proyeksi in didapat berdasarkan nilai yang ada. Jika bidang pandang diletakan pada posisi Z_{vp} sepanjang sumbu Z_v, maka setiap titik (x,y,z) didalam koordinat pandang ditraansformasikan kedalam koordinat proyeksi sebagai berikut

$$x_p = x,\ y_p = y$$

Pada proyeksi juga perlu didefinisikan sebuah ruang (*volume*) yang dimana bagian obyek yang terletak di luar ruang tidak ditampilkan pada layar.



Gambar 2.3 Proyeksi orthographic

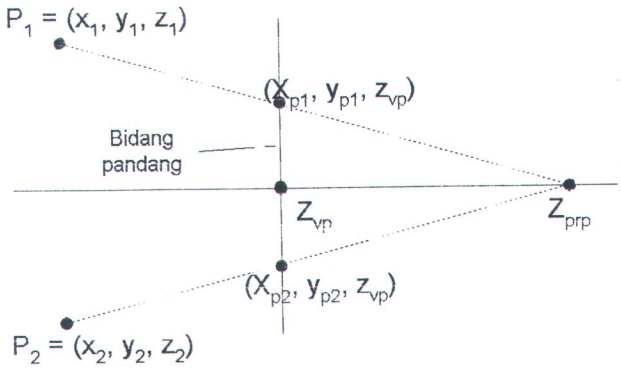
Untuk mendefinisikan proyeksi orthographic pada OpenGL menggunakan perintah `glOrtho()`. Perintah lengkapnya sebagai berikut :

```
void glOrtho(Gldouble left, Gldouble right, Gldouble bottom,
Gldouble top)
```

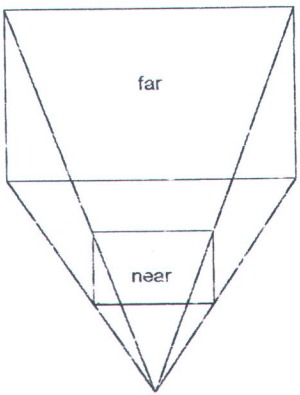
parameter dari perintah tersebut disesuaikan pada gambar 2.3.

2.1.3.2. Proyeksi Perspektif (*Perspective Projection*)

Untuk mendapatkan proyeksi persfektif dari sebuah obyek tiga dimensi. Semua titik-titik yang ada ditrasformasikan sepanjang garis proyeksi, dimana garis tersebut bertemu pada sebuah titik, yang disebut sebagai pusat transformasi. (*center of projection*, cp). Jika titik tersebut terletak pada Z_{cp} sepanjang sumbu Z_v, dan bidang pandang (*view plane*, vp) terletak pada Z_{vp}, seperti pada gambar .



Gambar 2.4 Proyeksi perspektif titik P_1 (x_1,y_1,z_1) ke posisi (x_{p1}, y_{p1}, z_{vp}) pada bidang pandang



Gambar 2.5 Proyeksi persfektif

Perintah `glFurstum()` pada OpenGL menghasilkan matriks proyeksi dan mengalikan matriks tersebut dengan matriks yang hasil transformasi sebelumnya. Perintah tersebut sebagai berikut :

$$R = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.2 Pewarnaan

Proses pewarnaan pada sebuah permukaan yang belangsung pada OpenGL didefiniskan oleh tiga unsur yaitu warna merah (R), warna hijau (H) dan biru(B). Proses pewarnaan yang diterangkan pada bagian ini, termasuk *Goraud Shading* dan juga *texturing*.

2.2.1. Pencahayaan

Ketika cahaya mengenai sebuah permukaan, maka energi dari cahaya tersebut dapat diserap, dipantulkan atau ditransmisikan. Beberapa energi tersebut diserap lalu dijadikan panas. Selebihnya dipantulkan atau ditransmisikan, dengan adanya sinar yang dipantulkan, membuat sebuah obyek dapat telihat. Jika semua energi cahaya diserap oleh obyek maka obyek tersebut menjadi tak terlihat, dan obyek tersebut dikenal sebagai obyek yang berwarna hitam.

untuk setiap arah. Sedangkan cahaya *specular* dipantulkan dan mempunyai arah yang tertentu, berdasarkan sudut datang.

Perhitungan intensitas cahaya yang dipantulkan, didasarkan pada hukum *Lambert's cosine* (*Lambert's cosine law*)[3]. Hukum ini menyatakan bahwa intensitas dari cahaya yang dipantulkan dari sebuah obyek adalah sebanding dengan cosinus sudut antara arah sinar dan normal vektor permukaan. Adapun persamaan matematis dinyatakan sebagai berikut :

$$I = I_i * k_d * \cos\theta \qquad (2\text{-}5)$$

dimana I adalah intensitas cahaya yang dipantulkan, I_i adalah intensitas dari sumber cahaya, k_d adalah konstanta pantulan yang bernilai antara 0 sampai 1, dan θ adalah sudut antara arah cahaya dan normal permukaan.

Perintah yang digunakan pada OpenGL untuk mendefinisikan besaran intesitas dari pantulan *diffuse* adalah dengan menggunakan perintah `glLight*()`.

```
void glLight{if}[v](Glenum light, Glenum pname, TYPE param);
```

Perintah ini akan membuat sumber cahaya *light*, yang dapat bernilai GL_LIGHT0, GL_LIGHT1,..., atau GL_LIGHT7. Nilai *pname* menunjukan parameter karakteristik dari cahaya tersebut, untuk pantulan diffuse nilai *pname* sama dengan GL_DIFFUSE. Sedangkan *param* berisikan nilai dari properti tersebut,

cahaya tersebut berwarna putih. Jika warna R, G dan B maka warna keseluruhan adalah putih tapi dengan intensitas yang setengahnya.

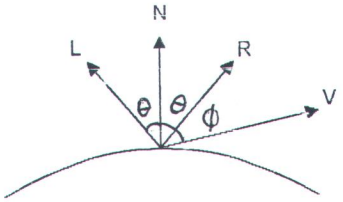
Pada kenyataan obyek juga menerima cahaya yang dipantulkan dari lingkungan sekitar. Cahaya ini dikenal dengan *ambient light*[4]. Dengan adanya cahaya tersebut maka persamaan diatas dapat dirubah menjadi :

$$I = I_i * k_d * \cos\theta + I_a * k_a \qquad (2\text{-}6)$$

dimana nilai I_a adalah intensitas cahaya ambient, dan k_a adalah konstanta ambient yang bernilai antara nol sampai dengan satu.

Untuk mendefiniskan unsur pantulan specular dan cahaya ambient, perintah yang dipakai pada OpenGL sama dengan perintah untuk mendefinisikan unsur pantulan diffuse, nilai *pname* yang dipakai adalah GL_SPECULAR, sedangkan untuk cahaya ambient nilai *pname* adalah GL_AMBIENT.

Pada gambar 2.6 diperlihatkan arah pantulan specular pada satu titik. Sudut pantulan specular sama dengan sinar datang, terdapat dua sudut pada sisi kanan dari vektor normal permukaan **N**. Pada gambar tersebut kita menggunakan **R** untuk merepresentasikan unit vektor yang menunujukan pantulan *specular*. **L** untuk merepresentasikan unit vektor dari sinar datang, dan **V** sebagai unit vektor



Gambar 2.6 Vektor pada pantulan specular[5]

Untuk perhitungan intensitas specular menggunakan *Phong specular-reflection*[6] dengan persamaan

$$I_{spec} = W(\theta) I_l \quad \cos^{ns} \phi \qquad (2\text{-}7)$$

dimana I_l adalah intesitas dari sinar sumber, dan W(0) adalah koefisien pantulan specular yang nilainya tergantung pada bahan permukaan tersebut.

Seperti cahaya, material mempunyai warna ambient, diffuse dan specular yang dipantulkan dari permukaan benda. Pantulan ambient dari material disatukan dengan pantulan ambient dari cahaya yang datang ke permukaan benda, pantulan diffuse material disatukan dengan pantulan diffuse cahaya, begitu juga dengan pantulan specular.

Sebelum pencahayaan pada OpenGL dapat dilakukan maka perlu diaktifkan sumber cahaya tersebut. Untuk mengaktifkan cahaya tersebut digunakan perintah `glEnable`(GL_LIGHTING), perintah ini mengaktifkan

GL_LIGHT0, maka yang perlu dilakukan adalah menambahakan perintah `glEnable`(GL_LIGHT0).

2.2.2. Gouraud Shading

Sebuah obyek dalam grafika komputer direpresentasikan dengan gabungan-gabungan poligon. Setiap poligon dapat direnderer dengan sebuah intensitas untuk keseluruhan daerah poligon, atau dengan intensitas berbeda-beda untuk tiap bagian dari poligon, intensitas tersebut dapat dicari berdasarkan setiap titik dari permukaan mengunakan interpolasi.

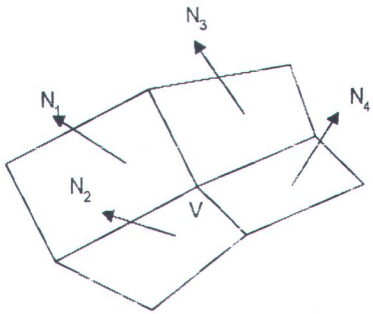
Memberikan warna sebuah poligon dengan liniear interpolasi intensitas (*linearly interpolation intenstiy*). Nilai intensitas untuk setiap poligon disamakan dengan nilai poligon yang disebelahnya.

Setiap permukaan poligon yang diwarnai dengan menggunakan Gouraud shading, dilakukan perhitungan seperti berikut :

- Tentukan rata-rata unit vektor normal untuk setiap poligon vertex.
- Lakukan model penchayaan kesetiap vertex untuk menghitung intensitas vertex tersebut.
- Lakukan interpolasi liniear pada tiap intensitas vertex sepanjang

$$Nv = \frac{\sum_{k=1}^{n} Nk}{\left| \sum_{k=1}^{n} Nk \right|} \qquad (2\text{-}8)$$

setelah didapat normal vektor, maka kita dapat menghitung intensitas padat vertex tersebut.



Gambar 2.7 Vektor normal untuk titik V

Langkah selanjut adalah menginterpolasi intensitas sepanjang garis tepi poligon. Untuk setiap *scan line*, intensitas pada titik pertemuan antara *scan line* dengan garis tepi poligon adalah interpolasi liniear dari intensitas 2 titik akhir dari garis tepi poligon.

Pada gambar 2.8 perhitungan untuk intensitas pada titik 4 dilakukan dengan cara :

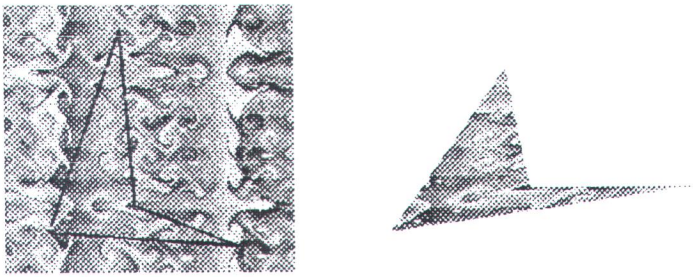
$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} * I_1 + \frac{y_1 - y_4}{y_1 - y_2} * I_2 \qquad (2\text{-}9)$$

2.2.3. Texture Mapping

Metode yang umum untuk menambahkan detail dari permukaan adalah dengan menambahkan pola tekstur ke dalam permukaan obyek tersebut. Pola texture ini dapat didefinisikan sebagai sebuah array persegi empat. Cara ini dikenalkan sebagai *texture mapping.*

Pola texture didefinisikan dengan jaring persegi panjang dari nilai intensitas dalam sebuah ruang texture (*texture space*), dengan sistem koordinat s dan t. Posisi permukaan dldalam gambar didefinisikan dengan sistem koordinat *u* dan *v*. Pemetaan Texture dapat dilakukan dengan dua cara. Pertama, kita dapat memetakan pola texture kedalam permukaan obyek, lalu kebidang proyeksi. Cara kedua, adalah memetakan kumpulan titik warna ke permukaan obyek lalu kebidang tetxure.

Dalam Gambar 2.9 diperlihatkan proses pemetaan tekstur pada sebuah



Gambar 2.9 Proses pemetaan tekstur[8]

Untuk melakukan proses pemetaan tekstur perlu didefinisikan tekstur pada OpenGL dengan perintah `glTexImage2D()` perintah ini membangun sebuah tekstur 2 dimensi. Perintah tersebut memerlukan beberapa argumen seperti yang terlihat dibawah ini :

```
void glTexImage2D(Glenum target, Glint level, Glint component,
Glsizei width, Glsizei height, Glint border, Glenum format, Glenum
type, const Glvoid *pixels);
```

Parameter dari *target* bernilaikan GL_TEXTURE_2D. Parameter *level* menunjukan tingkatan dari ukuran tekstur, dimana ukuran tersebut merupakan kelipatan dari pangkat 2. Parameter selanjutnya adalah *componets*, parameter ini akan bernilaikan 1 sampai 4, yang mengindekasikan komponen warna (R, G, B, dan A) tertentu yang akan digunakan pada proses *modulate* dan *blend*. Ukuran dari tekstur ditentukan oleh parameter *width* untuk lebar dan *height* untuk *tinggi*.

decal, modulate, blend. Untuk mendefinisikan fungsi ini pada OpenGL digunakan perintah `glTexEnv()`. Bentuk perintah tersebut sebagai berikut :

```
void glTexEnv{if}{v}(Glenum target, Glenum param, TYPE param)
```

nilai parameter *target* harus bernilaikan GL_TEXTURE_ENV. Sedangkan *pname* sama dengan GL_TEXTURE_ENV_MODE, parameter *param* dapat bernilaikan GL_DECCAL, GL_MODULATE, GL_BLEND.

Ketika tekstur divisualisasikan, koordinat dari obyek dan koordinat dari tekstur harus didefinisikan untuk setiap *vertex.* Setelah dilakukan transformasi, obyek koordinat menentukan posisi vertex pada layar, sedangkan tekstur koordinat menentukan *texel* (titik pada tekstur) mana yang ditempelkan pada *vertex.*

Perintah OpenGL untuk mendefinisikan koordinat tekstur adalah `glTexCoord()`. Dengan parameter sebagai berikut :

```
void glTexCoord{1234}{sifd}{v}(TYPEcoords)
```

dimana parameter coords menunujukan koordinat dari.

Sepertihalnya dengan pencahayaan dimana unsur cahaya harus diaktifkan terlebih dahulu, maka proses penteksturan juga harus diaktifkan terlebih dahulu dengan menggunakan perintah `glEnable`(GL_TEXTURE_2D).

BAB III

BAB III

FRAKTAL DAN TERRAIN MAP

Dalam bab ini dijelaskan tentang fraktal secara umum yang merupakan dasar dari pembuatan perangkat lunak, termasuk pembahasan mengenai dimensi dari sebuah obyek fraktal. Juga termasuk didalamnya bagaimana membangkitkan terrain map dengan metode fraktal. Di samping itu juga dibahas mengenai terrain map.

3.1. Fraktal

Obyek-obyek alam, seperti pegunungan dan awan mempunyai bentuk geometris yang tidak beraturan, yang sangat sulit untuk direpresentasika dengan menggunakan metode *Euclidean*. Obyek-obyek alam dapat dengan realistik dijabarkan dengan menggunakan metode fraktal, dimana prosedur yang digunakan untuk memodelkan suatu obyek dari pada persamaan matematis.

Sebuah obyek fraktal dihasilkan dengan melakukan fungsi transformasi tertentu pada satu titik di dalam area tertentu. Jika $P_0 = (x_0, y_0, z_0)$ merupakan titik awal, pada setiap iterasi sebuah fungsi transformasi F menghasilkan rincian

similarity dari sebuah obyek dapat dilihat dari berbagai bentuk, tergantung dari pada representasi dari obyek. Kita menjelaskan suatu obyek fraktal dengan sebuah prosedur yang mespesifikasikan operasi yang berulang untuk menghasilkan bagian-bagian yang lebih rinci dari suatu obyek. Obyek-obyek alam dipresentasikan dengan menggunakan prosedur yang secara teori dilakukan secara berulang.

Dalam gambar 3.1 diperlihatkan sebuah daun yang merupakan obyek alam, dimana jika dilihat secara rinci maka dapat disimpulkan bahwa bagian terkecil daari obyek tersebut merupakan pensekalaan dari bagian obyek secara keseluruhan. Atau dengan kata lain gambar 3.1 tersebut mempunyai sifat self-similarity.

Gambar 3.1 Obyek alam yang dibangun dengan metode fraktal[9]

3.1.1. Klasifikasi Fraktal[10]

Pembentukan obyek fraktal dibedakan atas tiga macam seperti berikut :

membuat skala s yang lebih kecil pada keseluruhan obyek. Selama proses iterasi dijalankan dapat digunakan faktor skala s yang sama untuk semua bagian atau dapat digunakan faktor skala yang berbeda untuk bagian.

- **Self-affine fractal**

  Mempunyai bagian-bagian yang dibentuk dengan menggunakan parameter skala yang berbeda s_x, s_y, s_z, pada arah koordinat yang berbeda. Seperti pada Self-similar selain itu dapat digunakan faktor skala yang sama untuk setiap iterasi, juga .dapat digunakan faktor skala yang berbeda. Terrain yang dibuat pada tugas akhir ini termasuk dalam kategori *self-affine*

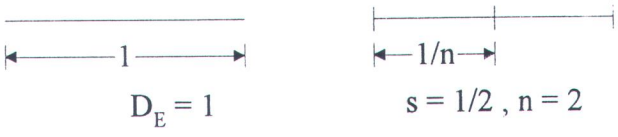
- **Invariant fractal sets**

  Dibentuk dengan transformasi nonlinear. Jenis fraktal ini termasuk *self-squaring*. *Self-squaring* menggunakan bilangan kompleks dengan persamaan $z = x + iy$, dengan x dan y adalah bilangan real, sedangkan $i^2 = -1$.

3.1.2. Dimensi Fraktal

Banyaknya variasi pada sebuah obyek fraktal dapat dijelaskan dengan menggunakan sebuah bilangan **D** atau yang disebut sebagai dimensi fraktal, yang merupakan suatu ukuran kekasaran atau fragment dari obyek tersebut. Semakin

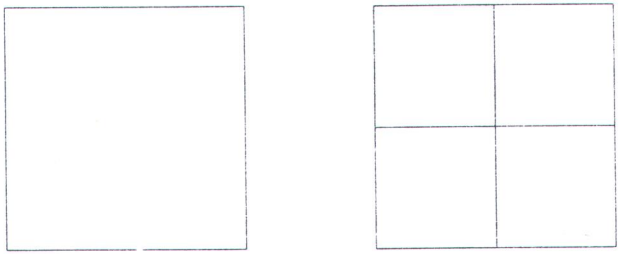
sebuah segment garis lurus, sebuah bujur sangkar, dan sebuah kubus. Dengan s = 1/2, garis lurus (gambar 3.2 (a)) dibagi menjadi 2 bagian sama panjang. Sebuah bujur sangkar pada gambar 3.2(b) dibagi menjadi menjadi empat bagian yang sama besar. Dan sebuah kubus dibagi menjadi delapan bagian yang sama besar.



$D_E = 1$ $\quad$ $s = 1/2 , n = 2$

$s^1 = 1/n$

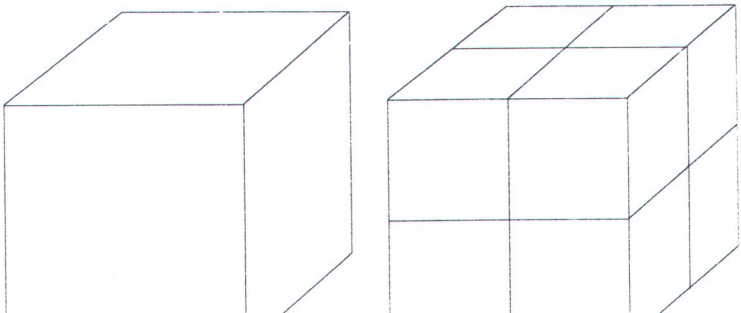
(a)



$D_E = 2$ $\quad$ $s = 1/2, n = 4$

$s^2 = 1/n$

(b)



Untuk setiap obyek tersebut, hubungan antara jumlah sub bagian dan faktor skala adalah $n.s^D = 1$. Dalam analogi dengan obyek *Euclidean*, dimensi fraktal D untuk obyek *self-semilar* didapat dari

$$s^D = 1/n \tag{3.1}$$

Dari persamaan diatas didapat nilai D sebagai dimensi fraktal self-similarity sebagai berikut :

$$D = \frac{\ln n}{\ln(1/s)} \tag{3.2}$$

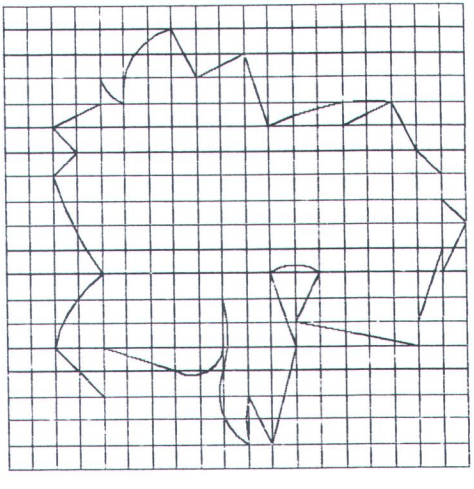
untuk sebuah self-similar fraktal yang dibangun dengan faktor skala yang berbeda untuk bagian-bagian yang berbeda, dimensi dari fractal semilarity didapat dengan cara

$$\sum_{k=1}^{n} s_k^D = 1 \tag{3.3}$$

dimana s_k adalah faktor skala untuk bagian yang ke-k..

Jika obyek yang dibentuk mempunyai mempunyai berbagai macam variasi, tidak hanya sebuah garis lurus ataupun kubus, untuk menentukan dimensi dari obyek tersebut membutuhkan ketelitian tertentu. Salah satu cara ialah

Dimensi fraktal dari sebuah obyek dapat ditentukan dengan metode *box-covering* menggunakan bujur sangkar, seperti dalam gambar 3.3. Area di dalam bentuk yang tak beraturan dapat ditentukan dengan menjumlahkan area dari bujura sangkar yang menuntupinya.



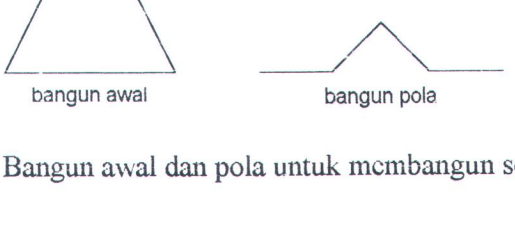
Gambar 3.3 Box-covering pada sebuah bentuk tak beraturan

Metode *box-covering* diaplikasikan dengan menenentukan koordinat paling luar dari obyek tersebut, lalu obyek dibagi menjadi sejumlah kotak yang kecil dengan faktor skala tertentu. Jumlah kotak yang menutupi obyek tersebut dinamakan *box-dimension,* dan n dihubungkan dengan dimensi fraktal dari obyek tersebut. Untuk menentukan dimensi fraktal digunakan persamaan 3.2. Untuk

permukaan *Euclidean* adalah 2 dimensi, dan benda padat *Euclidean* mempunyai dimensi 3.

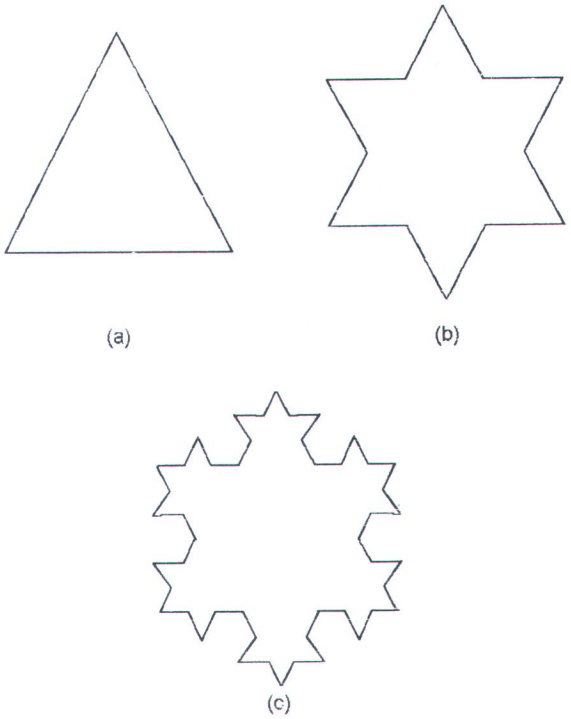
Untuk sebuah kurva fraktal terletak di dalam sebuah bidang dua dimensi, dimensi fraktalnya (D) lebih besar dari 1 (dimensi dari kurva *Ecludian* adalah 1). Semakin mendekati nilai 1, maka kurva tersebut semakin halus,. Jika D=2 maka kurva tersebut memenuhi sebuah ruang dua dimensi. Fraktal permukaan mempunyai nilai dimensi antara $2 < D <= 3$.

Sebagai contoh, untuk membangun fraktal yang *self-similar*, diperlukan sebuah bangun awal dan sebuah bentuk pembangkit atau pola. Seperti pada gambar 3.3 di mana terdapat bentuk bangun awal dan bentuk pola.



Gambar 3.4 Bangun awal dan pola untuk membangun sebuah fraktal

Dengan pola bangun seperti pada gambar 3.4 dapat dibangun fraktal pada



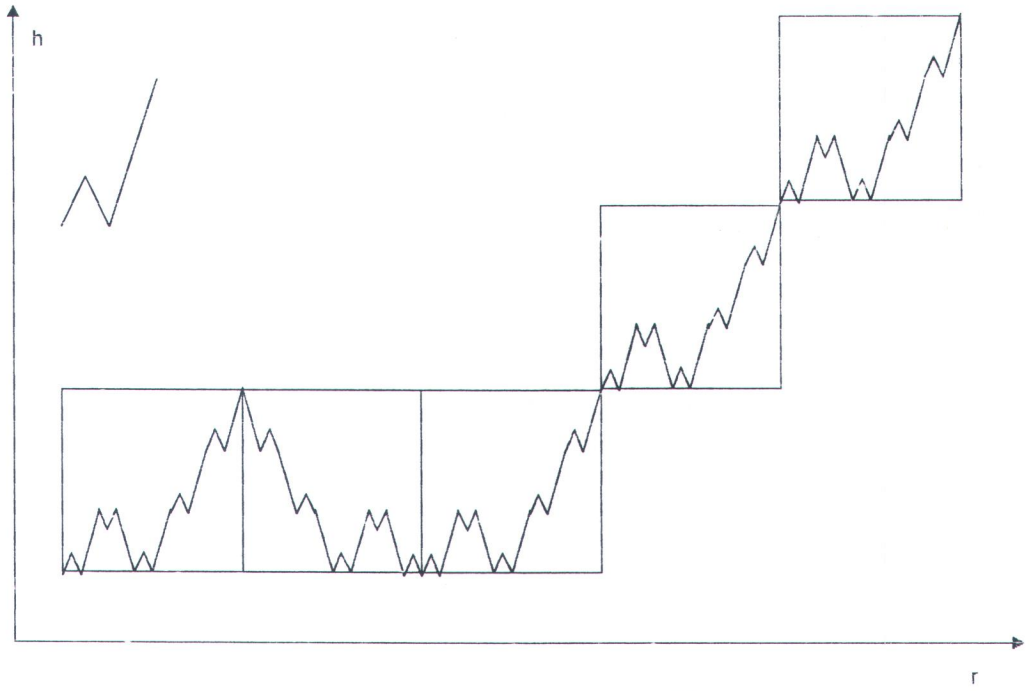
Gambar 3.5. Dua iterasi pertama ((b) dan (c)) dari sebuah obyek fraktal.

3.1.3. Fraktal Permukaan

Sebuah permukaan dapat didefiniskan dengan sebuah fungsi h(r), yang merupakan suatu fungsi ketinggian dari permukaan pada koordinat r(x,y) pada sebuah bidang datar.

Untuk kebanyakan permukaan, fungsi h(r) adalah karakteristik dari sebuah fraktal *self-affine*. Terdapat sebuah variabel exponent H yang bernilai kurang dari

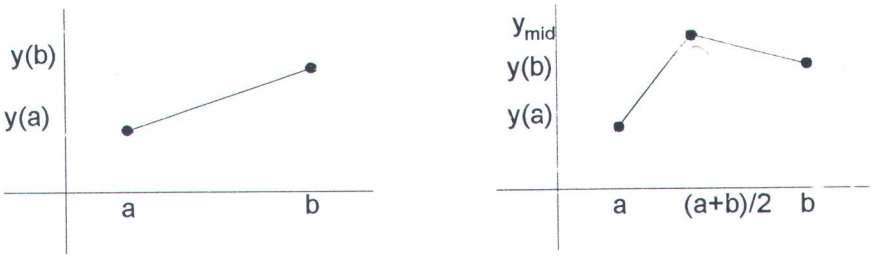
$$d_f = d - H \qquad (3.4)$$



Gambar 3.6 Self-affine fraktal [12]

Pada contoh yang ditunjukan pada gambar 3.6 dapat diketahui transforamasi yang dilakukan adalah r -> 5r dan h -> 3h, maka nilai H = log(3)/log(5) = 0.68 dan d_f = 1.32. dimensi d -i =1.

Metode random midpoint displacement merupakan salah satu metode yang digunakan untuk menghasilkan suatu *terrain.* Gambar 3.7 menunjukkan metode tersebut yang diterapkan pada sebuah garis lurus bidang xy. Dimulai dengan



Gambar 3.7 Proses midpoint displacement pada sebuah garis

Nilai r didapat berdasarkan distribusi Gaussian (*Gaussian Distribution*) dengan mean sama dengan nol, yang dikalikan dengan nilai ratio, dimana nilai ratio tersebut didapat dari 2^{-H} pada setiap iterasi nilai ratio dikalikan dengan nilai ratio itu sendiri.

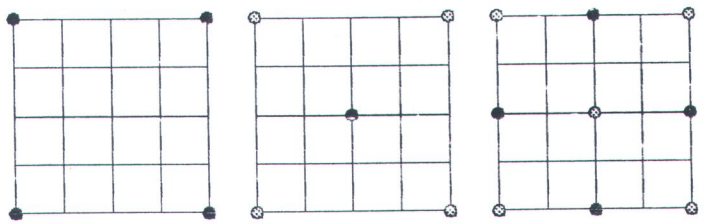
Pembuatan *terrain* dengan menggunakan metode random midpoint-displacement adalah dengan mengimplementasikan prosedur tersebut pada sebuah bidang datar seperti pada gambar 3.8. Dimulai dengan memberikan sebuah nilai ketiggian z untuk setiap sudut (a, b, c, dan f dalam gambar 3.8) pada bidang datar tersebut. Selanjutnya dicari untuk masing-masing nilai pada titik k,e,f,g dan h. Sebagai contoh untuk mencari nilai ketinggian pada titik m dicari dengan persamaan

$$z_k = (z_a + z_b + z_c + z_d) / 4 + r_k$$

sedangkan untuk titik e, f, g dan h dapat dicari dengan mengunakan titik k.

Pada gambar 3.9 menunjukkan proses *midpoint-displacement* yang dilakukan sebanyak 2 kali interasi untuk tiap-tiap satu iterasi dilakukan 2 kali langkah, yang dikenal dengan langkah *diamond* dan *square*. Pada gambar 3.9(a). merupakan langkah awal dengan memberikan nilai random untuk masing-masing titik yang berwarnakan hitam. Selanjutnya dicari titik tengah seperti pada gambar 3.9(b), dengan nilai rata-rata dari keempat titik yang terdapat pada sudut-sudut. Sedangkan gambar 3.9(c), 3.9(d), dan 3.9(e) merupakan proses kelanjutan untuk mencari nilai ketinggian tiap titik yang diberi warna hitam berdasarkan nilai rata-rata keempat titik yang berwarna putih.

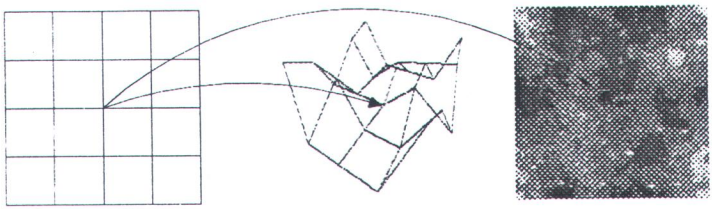
Dimensi dari permukaan dapat ditentukan dengan D = 3-H. Dimana H adalah suatu nilai *exponent* dan D adalah dimensi permukaan yang bernilai lebih besar dari 2. Dari nilai H kita dapat mencari faktor skala f dengan persamaan $f = 2^{(-H)}$, yang digunakan untuk mereduksi hasil nilai random r.



(a) (b) (c)

3.2. Terrain Maps

Suatu terrain map didefinisikan sebagai suatu fungsi yang memetakan koordinat dua dimensi (x,y) kepada suatu nilai ketinggian *a* dan warna *c*. Dengan kata lain, suatu terrain map adalah suatu fungsi yang menjelaskan topografi suatu daerah. Gambar 3.10 memperlihatkan suatu proses pembuatan sebuah terrain, dimana nilai x dan y, bernilai dari 0 sampai 1



Gambar 3.10 Terrain Map

Pada tugas akhir ini nilai warna dimasukan oleh pemakai, sedangkan ketinggian didapatkan dengan metode fraktal yang telah dijelaskan pada bagian sebelumnya.

BAB IV

BAB IV

PERANCANGAN DAN PEMBUATAN

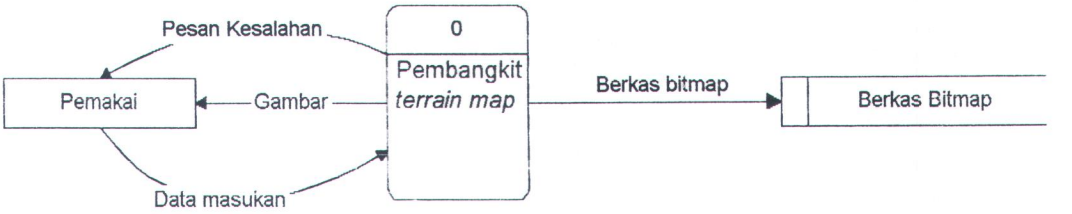
PERANGKAT LUNAK

Dalam Bab ini dijelaskan bagaimana perangkat lunak dikembangkan. Penjelasan meliputi kebutuhan sistem, deskripsi sistem, perancangan dan implementasi dari perancangan tersebut. Perangkat lunak yang dikembangkan dibuat menggunakan bahasa C, dalam hal ini *Microsoft Visual C++ ver. 6*, yang dijalankan pada lingkungan sistem operasi *windows*.

4.1 KEBUTUHAN SISTEM

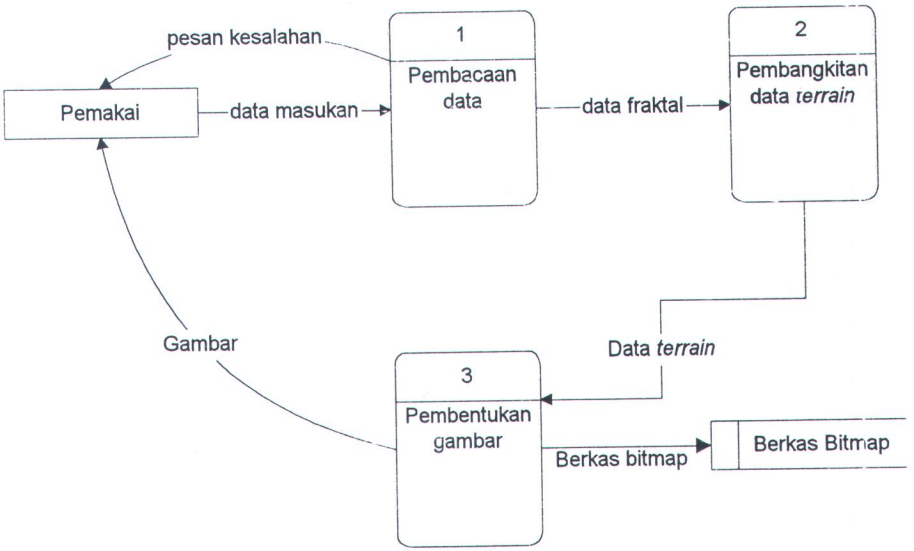
Perangkat lunak yang dibuat merupakan aplikasi yang hanya dapat dijalankan dalam lingkungan sistem operasi *windows*, dalam hal ini versi sistem operasi *windows* yang digunakan adalah versi '95 dan '98. Selain itu untuk menjalankan perangkat lunak tersebut memerlukan *run-time library* berupa file dll, baik yang disediakan oleh *Microsoft visual C++*, maupun *run-time library* yang disediakan oleh OpenGL.

yang dibuat. Dari gambar dapat dilihat aplikasi menerima masukan dari pemakai berupa data-data yang dibutuhkan untuk membangkitkan suatu permukaan, seperti jumlah iterasi yang akan dilakukan. Keluaran dari perangkat lunak merupakan gambar pada layar komputer maupun alat cetak, selain itu juga menghasilkan berkas gambar *bitmap*. Perangkat lunak juga akan menampilkan pesan kesalahan dari masukan pemakai.



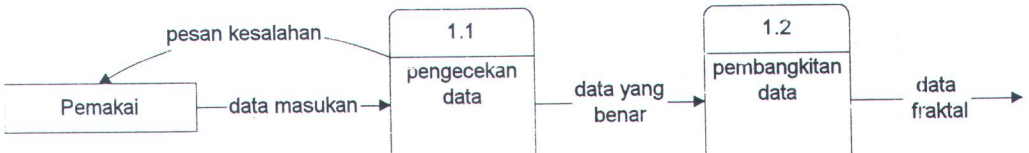
Gambar 4.1 DFD tingkat 0, Aplikasi Pembangkit terrain map.

Gambar 4.2 merupakan penjelesan lebih rinci mengenai proses yang dijalankan pada aplikasi. Proses tersebut diuraikan dalam bentuk DFD tingkat 1. Proses pembagkitan terrain map terdiri dari tiga proses utama, yaitu pembacaan data, pembangkitan data terrain, dan pembentukan gambar. Proses pembacaan data menghasilkan data yang diperlukan untuk membangkitakan data terrain map, seperti banyaknya iterasi yang akan dilakukan dan nilai dimensi dari fraktal. Proses pembangkitan merupakan proses inti dimana akan dihasilkan nilai

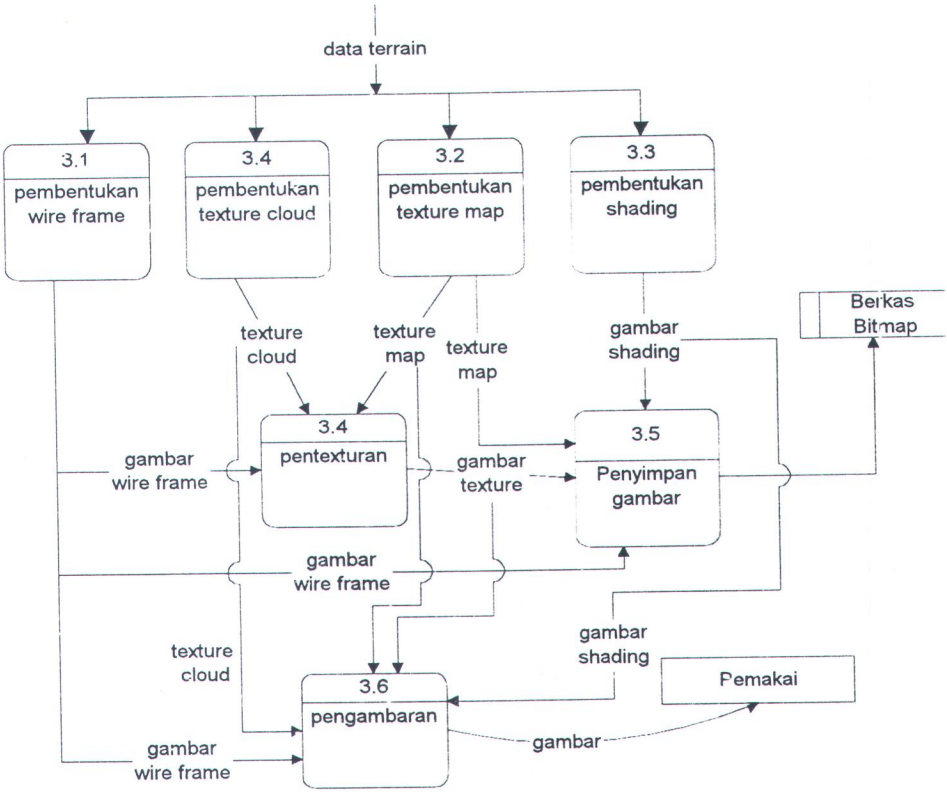


Gambar 4.2 DFD tingkat 1 Pembangkitan terrain map

Pada proses pembacaan data dilakukan pengecakaan data masukan pemakai. Pengecekan yang dilakukan berupa batasan nilai minimum dan maximum suatu nilai masukan pemakai, juga nilai floating point dari masukan tersebut. Selanjutnya dilakukan proses pembangkitan data fraktal, yang digunakan untuk membangkitkan data fraktal, dan juga dilakukan proses inisialisasi dari masukan tersebut. Gambar 4.3 menunjukan alur data dan proses yang lebih rinci, dalam pembacaan data.



Proses pembentukan gambar dapat dilihat pada gambar 4.4. Proses ini dibagi menjadi 6 proses yang lebih kecil yaitu pembentukan wire frame, pembentukan texture, pembentukan shading, pentexturan, pengambaran dan penyimpanan gambar. Kesemua proses tersebut, kecuali pentekstur an, pengambaran dan penyimpanan gambar, menerima masukan berupa data permukaan. Pembentukan wire frame, merupakan proses pembentukan gambar rangka yang menghubungkan titik-titik ketinggian dari sebuah permukaan. Pembentukan texture, merupakan proses yang berfungsi untuk membentuk gambar texture berdasarkan data ketinggian, berdasarkan data ketinggian akan ditentukan warna apa yang akan diberikan untuk ketiggian tersebut. Dalam hal ini ketiggian dibagi 3 daerah, daerah sepertiga pertama diberi warna pertama, sepertiga kedua diberi warna kedua, dan sepertiga selebihnya diberi warna ketiga. Proses pembentukan shading, merupakan proses pemberian warna, seperti pada proses pembentukan texture map, hanya saja yang membedakan adalah cara penampilan gambar hasil, pada pembentukan texture map gambar yang dihasilkan dilihat dari atas, sedangkan pada proses pembentukan shading gambar dilihat secara persfektif. Pentekstur an adalah memberikan nilai warna untuk wire frame yang dihasilkan, dengan warna yang dihasilkan pada proses pembentukan texture



Gambar 4.4 DFD tingkat 2, pembentukan gambar

4.3 PERANCANGAN DAN IMPLEMENTASI PERANGKAT LUNAK

Pada bagian ini dijelaskan mengenai perancangan dan implementasi sistem perangkat lunak. Perancangan tersebut meliputi spesifikasi data dan pendekatan yang digunakan dalam sistem perangkat lunak. Sedangkan implementasi dari perangkat lunak, juga diterangkan dalam bentuk bahasa

terrain. Untuk tiap-tiap model yang dihasilkan, model permukaan, model shading dan awan mempunyai nilai kekasaran (H) dan iterasi yang berbeda.

Selain itu ketinggian dari air juga dimasukan oleh pemakai, nilai ini menunjukan bahwa permukaan yang dibentuk, pada ketinggian di bawah ketinggian air akan disamakan dengan ketinggian air

4.3.2 Perancangan Data Keluaran

Hasil dari perangkat lunak, adalah data gambar baik yang ditampilkan ke layar komputer, ke alat cetak, ataupun disimpan dalam berkas. Penyimpanan data ke dalam berkas dalam format BMP.

4.3.3 Perancangan Data Saat Pemrosesan

Selain data masukan pemakai dan data yang dihasilkan oleh perangkat lunak, terdapat data yang diperlukan oleh perangkat lunak untuk melakukan proses pembangkitan terrain, ada pun data tersebut adalah :

1. Data nilai ketinggian untuk permukaan, untuk proses shading, dan data untuk pembentukan awan. Data ini dimasukan dalam array satu dimensi yang jumlahnya tergantung pada masukan pemakai.
2. Data properti dari cahaya, berupa nilai data ketentuan cahaya berupa nilai

4.3.4 Implementasi Struktur data

Dalam bagian ini dijelaskan mengenai implementasi struktur data digunakan pada pembuatan perangkat lunak. Kelas CFractal digunakan untuk menyimpan data fraktal, ketinggian, dan juga operasi pencarian data-data ketinggian. Data ketinggian disimpan dalam array *fractal_arr* dengan tipe data *float*. Seperti yang terlihat pada bagian di bawah ini.

```
class CFractal
{
public:

      int get_size();
      void FindMinMax(float *max, float *min);
      float min;
      float max;
      float get_array(int i);
      void Redesign(int iter, int dim, float h, int seed);
      float get_array(int i, int j);
      float gauss_dist(float min, float max);
      float get_Squareavg(int i, int j, int step);
      float get_Diamondavg(int i, int j, int step);
      float TwoPointAvg(int a, int b);
      void FillArrayTwoDim(float heightScale);
      void FillArrayOneDim(float heightScale);
      float fractal_dim;
      void freeFractalArray();
      void allocFractalArray();
      Fractal();
      void fillFractalArray(int seed, float heightScale);
      virtual ~Fractal();
      float *fractal_arr;

protected:
      int dimension;
```

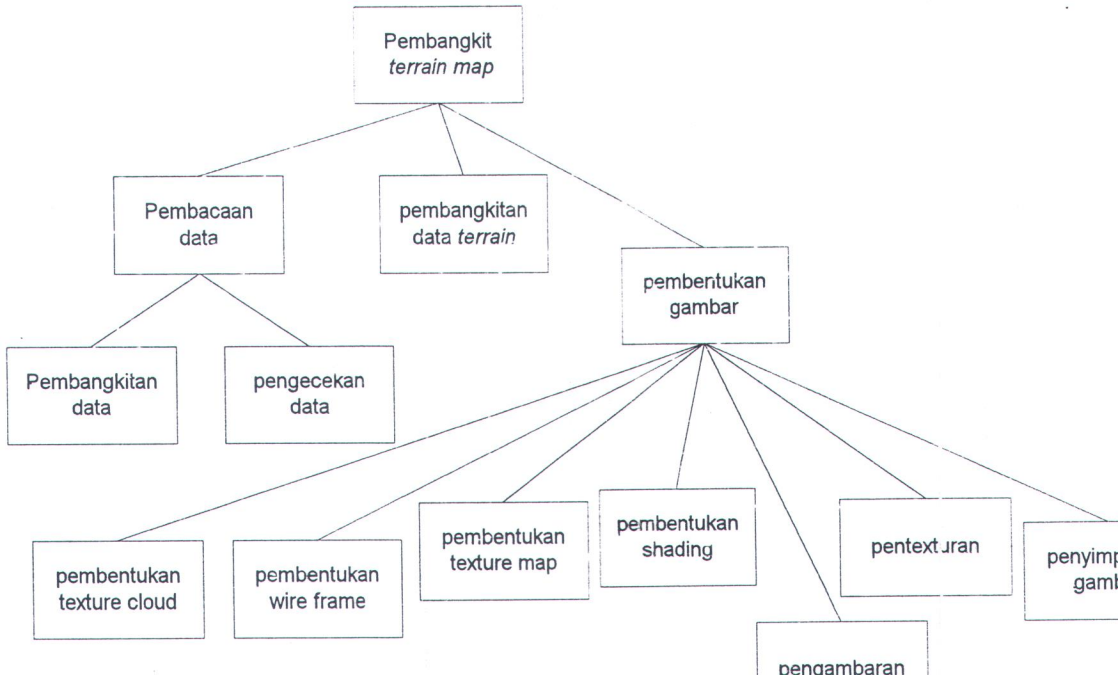
```
class CfractalTerrainView ..........
{

protected: // create from serialization only
      CFractalTerrainView();
      DECLARE_DYNCREATE(CfractalTerrainView)

// Attributes
public:
      CFractal Land,Texture,Cloud,Mine;
      Float Texture_H, Land_H, Cloud_H;

      GLfloat Trans_z;
      GLfloat Trans_y;
      GLfloat Rot_y;

      COLORREF Color1,Color2,Color3;

      BOOL shiftKey;
      BOOL water_level ;

      UINT Texture_I;
      UINT Cloud_I;
      UINT Land_I;

      UINT tile;
      UINT rndmSeed;

      virtual void OnSize (UINT nType, int cx, int cy);

      float aspectRatio;
      int Height_win;
      int Width_win;
      BOOL drawShading();
      BOOL drawTexture ();
      BOOL drawAllScene ();
      BOOL drawTriangle ();

      CfractalTerrainDoc* GetDocument();

// Operations
public:
```

```
// Implementation
public:
      float water_height;
      void initLights();
      void drawlTileLine();
      void drawLine();
      void drawlPlane();
      void FindColorHeight(float *f,int s, float *h1, float *h2,
float *water, int wl);
      void FindCloudColor(int size, GLubyte *CloudColor);
      void FindColor(float h, float h1,float h2,float w, int lw);
      void FindNormal(float x1, float y1, float z1,
                              float x2, float y2, float z2,
                              float x3, float y3, float z3);
      BOOL drawWireFrame();
      BOOL drawCloud();
      void fillpoly(float *f, int sz,int light, int teks,int wl);
      virtual ~CfractalTerrainView();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
      //{{AFX_MSG(CFractalTerrainView)
      afx_msg void OnViewOptions();
      afx_msg void OnViewLine();
      afx_msg void OnViewUpdateLine(CCmdUI* pCmdUI);
      afx_msg void OnViewMesh();
      afx_msg void OnViewUpdateMesh(CCmdUI* pCmdUI);
      afx_msg void OnViewTriangle();
      afx_msg void OnViewUpdateTriangle(CCmdUI* pCmdUI);
      afx_msg void OnViewCloud();
      afx_msg void OnViewUpdateCloud(CCmdUI* pCmdUI);
      afx_msg void OnViewShading();
      afx_msg void OnViewUpdateShading(CCmdUI* pCmdUI);
      afx_msg void OnViewTexture();
      afx_msg void OnViewUpdateTexture(CCmdUI* pCmdUI);
      afx_msg void OnViewRender();
```

```
        RenderType render_type;
        BOOL SceneIs_3D;
;
```

4.3.5 Hirarki Proses dan Implementasinya

Bagian ini menjelaskan proses-proses yang ada, dan implementasi dari proses tersebut berdasarkan perangkat lunak. Hirarki dari proses ditunjukan pada gambar 4.5, berdasarkan DFD yang telah dijelaskan sebelumnya. Implementasi dari proses disajikan berdasarkan bahasa pemrograman yang dipakai untuk membuat aplikasi.



pengecekan data dilakukan oleh prosedur *DoDataExchange()* dari CFractalOptionDialog. Dan proses pembangkitan data dilakukan oleh prosedur OnOption() dari kelas CfractalTerrainView.

```
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CFractalOptionsDlg)
        DDX_Control(pDX, IDC_COLOR_3, m_Color_3);
        DDX_Control(pDX, IDC_COLOR_2, m_Color_2);
        DDX_Control(pDX, IDC_COLOR_1, m_Color_1);
        DDX_Text(pDX, IDC_LAND_I, m_land_I);
        DDV_MinMaxUInt(pDX, m_land_I, 1, 10);
        DDX_CBString(pDX, IDC_RENDER_TYPE, m_renderType);
        DDV_MaxChars(pDX, m_renderType, 25);
        DDX_Text(pDX, IDC_RANDOMSEED, m_randomSeed);
        DDV_MinMaxUInt(pDX, m_randomSeed, 0, 32767);
        DDX_Text(pDX, IDC_TILE, m_tile);
        DDV_MinMaxUInt(pDX, m_tile, 1, 10);
        DDX_Check(pDX, IDC_INVERT, m_invert);
        DDX_Text(pDX, IDC_F_CLOUD_I, m_cloud_I);
        DDV_MinMaxUInt(pDX, m_cloud_I, 1, 10);
        DDX_Text(pDX, IDC_TEXTURE_ITERATIONS, m_texture_I);
        DDV_MinMaxUInt(pDX, m_texture_I, 1, 10);
        DDX_Text(pDX, ID_CLOUD_H, m_cloudH);
        DDV_MinMaxFloat(pDX, m_cloudH, 0.f, 1.f);
        DDX_Text(pDX, ID_LAND_H, m_land_H);
        DDV_MinMaxFloat(pDX, m_land_H, 0.f, 1.f);
        DDX_Text(pDX, ID_TEXTURE_H, m_texture_H);
        DDV_MinMaxFloat(pDX, m_texture_H, 0.f, 1.f);
        DDX_Check(pDX, IDC_WATER_LEVEL, m_water_level);
        DDX_Text(pDX, IDC_WATER_HEIGHT, m_water_height);
        DDV_MinMaxFloat(pDX, m_water_height, 0.f, 1.f);
        //}}AFX_DATA_MAP
//      DDX_Control(pDX, IDC_COLOUR1, m_ColourBox);
```

- Proses pembangkitan data Terrain

  Proses ini merupakan proses utama dari dari aplikasi yang dibuat. Dalam

```
    factal_arr  = ((float *) malloc (sizeof(float) * size * size));
}
```

Operasi pertama yang dilakukan adalah freeFractalArray(), proses ini berfungsi untuk menghapus data-data yang lama, lalu diikuti dengan proses allocFractalArray() yang berguna untuk mengalokasikan data sebanyak kebutuhan. Selanjutnya adalah fillFractalArray() berguna untuk mendapatkan nilai-nilai ketinggian yang baru, banyaknya jumlah data ketinggian didapat berdasarkan masukan pemakai, seperti yang dijelaskan pada **BAB III** mengenai **Fraktal dan Terrain Maps**. Selama proses proses fillFractalArray() didalamnya dijalankan proses get_Squareavg() dan get_Diamondavg().

```
.. ..
subSize = size - 1;
ratio = (float) (pow (2.f,-H));
scale = Scale * ratio;
step = subSize / 2;
fractal_arr[0] =
fractal_arr[(subSize*size)] =
fractal_arr[(subSize*size)+subSize] =
fractal_arr[subSize] = 0.f;
while (step) {
      for (i=step; i<subSize; i+=step) {
            for (j=step; j<subSize; j+=step) {
                  tmp = gauss_dist(-0.5f,0.5f) * scale +
                              get_Squareavg (i, j, step);
                  fractal_arr[(i * size) + j] = tmp;
                  j += step;
            }
            i += step;
      }
```

.. .. ..

Seperti yang telah dijelaskan pada bab sebelumnya algoritma midpoint-displacement dibagi menjadi dua langkah yaitu *square* dan *diamond.* Fungsi get_Squareavg() dan get_Diamondavg(), digunakan untuk melakukan dua operasi tersebut. Pada pengambilan data untuk langkah *diamond* perlu diperhatikan pada kondisi dimana titik yang akan dihasilkan berada pada tepi garis.

```
float CFractal::get_Diamondavg(int i, int j, int step){
if (i == 0)
      return ((float) (fractal_arr[(i*size) + j-step] +
                  fractal_arr[(i*size) + j+step] +
                  fractal_arr[((size-step-1)*size) + j] +
                  fractal_arr[((i+step)*size) + j]) * 0.25f);
else if (j == 0)
      return ((float) (fractal_arr[((i-step)*size) + j] +
                  fractal_arr[((i+step)*size) + j] +
                  fractal_arr[(i*size)+ j+step] +
                  fractal_arr[(i*size)+size-1-step])* 0.25f);
else if (i == size-1)
      return ((float) (fractal_arr[(i*size) + j-step] +
                  fractal_arr[(I*size) + j+step] +
                  fractal_arr[((i-step)*size) + j] +
                  fractal_arr[((step)*size) + j]) * 0.25f);
else if (j == size-1)
      return ((float) (fractal_arr[((i-step)*size) + j] +
                  fractal_arr[((i+step)*size) + j] +
                  fractal_arr[(i*size) + j-step] +
                  fractal_arr[(i*size) + 0+step]) * 0.25f);
else
      return ((float) (fractal_arr[((i-step)*size) + j] +
                  fractal_arr[((i+step)*size) + j] +
                  fractal_arr[(i*size) + j-step] +
                  fractal_arr[(i*size) + j+step]) * 0.25f);
}
```

Pada proses ini terbagi menjadi beberapa proses seperti yang terlihat pada gambar 4.5. yang masing-masing dilakukan dengan prosedur tersendiri. Pembentukan wire frame dilakukan oleh prosedur drawWireFrame(). Proses pembentukan texture dilakukan oleh prosedur drawTexture(). Sedangkan proses pembentukan shading dilakukan pada proses drawShading(). Proses pembentukan awan dilakukan pada prosedur drawCloud(). Proses pentexturan dilakukan pada prosedur drawRender(). Dan proses penyimpanan gambar dilakukan pada prosedur OnFileSave (). Sedangkan proses pengambaran dilakukan pada prosedur RenderScene().

Langkah yang dilakukan pada prosedur RenderScene () dapat dilihat di bawah ini.

```
.. .. ..
if (SceneIs_3D&&(render_type == Render_mesh)){
      return drawWireFrame();
}
if (render_type == Render_cloud)
      return (drawCloud());
else if (render_type == Render_shading)
      return (drawShading ());
else if (render_type == Render_triangle)
      return (renderTriangle ());
else if (render_type == Render_texture)
      return (drawTexture());
else if (render_type == Render_render)
      return (drawAllScene ());
drawLine();
```

```
for (i=0; i<size; i++) {
      x = -1.f;
      for (j=0; j<subSize; j++) {

            y1 = Land.get_array (i,j);
            y2 = Land.get_array (i,j+1);
            if(water_level){
                  if (y1 < h) y1 = h;
                  if (y2 < h) y2 = h;
            }

            glBegin (GL_LINES);
            glVertex3f (x, y1 , z);
            glVertex3f (x+inc, y2, z);
            glEnd ();

            if (i<subSize){
                  y1 = Land.get_array (i,j);
                  y2 = Land.get_array (i+1,j)
                  if (water_level) {
                        if (y1 < h) y1 = h;
                        if (y2 < h) y2 = h;
                  }
                  glBegin (GL_LINES);
                  glVertex3f (x, y1 , z);
                  glVertex3f (x, y2, z+inc);
                  glEnd ();
            }
          x += inc;
      }
      if (i<subSize){

            y1 = Land.get_array (i,j);
            y2 = Land.get_array (i+1,j)

            if(level_water){
                  if ( y1 < h) y1 = h;
                  if ( y2 < h) y2 = h;
            }
            glBegin (GL_LINES);
            glVertex3f (x, y1 , z);
            glVertex3f (x, y2, z+inc);
```

```
Void CfractalTerrainView:: initLights (){
    .....
    GlLightfv (GL_LIGHT1, GL_DIFFUSE, light_diffuse);
    GlLightfv (GL_LIGHT1, GL_SPECULAR, light_specular);
    GlLightfv (GL_LIGHT1, GL_POSITION, light_position);

    GlLightModelfv (GL_LIGHT_MODEL_AMBIENT, light_ambient);
    GlEnable (GL_LIGHT1);
    GlColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
    GlEnable (GL_COLOR_MATERIAL);
    ......
}
```

Sedangkan untuk proses pembuatan shading pada prosedur drawShading(). Pada prosedur ini dilakukan pembagian tingkatan menjadi tiga, guna memberikan warna untuk tiap-tiap bagian. Proses tersebut diperlihatkan pada bagian di bawah ini.

```
initLights ();
glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
glEnable (GL_DEPTH_TEST);
glEnable (GL_LIGHTING);
glClear (GL_COLOR_BUFFER_BIT);
FindColorHeight(f, Size_I, &h1, &h2, &water, water_level);
for (i=0; i<subSize; i++) {
    glBegin (GL_TRIANGLE_STRIP);
      x = -1.0f, j=0;
      y  = Texture.get_array (i,j);
      y1 = Texture.get_array (i+1,j)
      y2 = Texture.get_array (i,j+1);

      if (water_level) {
            if (y < water) y = water;
            if (y1 < water) y1 = water;
            if (y2 < water) y2 = water;
      }
      FindNormal (x,y,z,x,y1, z+inc,  x+inc, y2, z);
```

```
        FindColor (y, h1, h2,water,water_level);
        glVertex3f (x, y, z+inc);

        for (j=0; j<subSize; j++) {

                y  = Texture.get_array (i,j);
                y1 = Texture.get_array (i+1,j);
                y2 = Texture.get_array (i,j+1);

                if (water_level) {
                        if (y < water) y = water;
                        if (y1 < water) y1 = water;
                        if (y2 < water) y2 = water;
                }

                FindNormal(x,y, z, x, y1, z+inc,  x+inc, y2, z);
                FindColor (y2, h1, h2,water, water_level);
                glVertex3f (x+inc, y2, z);

                y  = Texture.get_array (i+1,j);
                y1 = Texture.get_array (i+1,j+1);
                y2 = Texture.get_array (i,j+1);

                if (water_level) {
                        if (y < water) y = water;
                        if (y1 < water) y1 = water;
                        if (y2 < water) y2 = water;
                }

                FindNormal(x,y,z+inc,x+inc,y1,z+inc, x+inc, y2, z);
                FindColor (y1, h1, h2,water, level_ water);
                glVertex3f (x+inc, y1, z+inc);
                x += inc;
        }
        z += inc;
        glEnd ();
}
glFlush ();
```

Dalam pembuatan tekstur sama dengan pembentukan shading, hal yang membedakan adalah pada proses pembentukan tekstur gambar dilihat dari atas

batasan tertentu diubah menjadi putih sedangkan dibawah nilai rata-rata ketinggian ditambah batasan yang sama, diubah menjadi biru, sedangkan nilai diantara batasan tersebut akan mengalami degradasi warna dari putih ke biru sesuai ketinggian tersebut. Berikut proses yang mengubah nilai ketinggian menjadi warna pada clouds.

```
.. .. ..
for (i=0; i<size*size; i++) avarage += cloud.get_array (I);
avarage = (avarage / (float)(size*size));

for (i=0; i<size-1; i++) {
      for (j=0; j<size-1; j++) {
            y = cloud.get_array (i,j);
            if (y<(avarage - range))       sky_shade = -range;
            else if (y>(avarage + range)) sky_shade = range;
            else sky_shade = y - avarage;

            sky_shade = ((sky_shade + range) * 1.25);
            cloud_tex[0]= (Glubyte)((0.4 + sky_shade) * 255);
            cloud_tex[1]= (Glubyte)((0.4 + sky_shade) * 255);
            cloud_tex[2]= (Glubyte)(0.9 * 255.);
            cloud_tex += 3;
      }
 }
.......
```

Proses pembentukan gambar yang lainnya adalah menggabungkan keseluruhan bagian, yaitu gambar cloud dan tekstur yang didapat dari proses sebelumnya. Berikut ini bagian program untuk menjalankan pembentukan keseluruhan gabungan gambar tersebut.

```
glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,(GLfloat)
```

```
 glTexCoord2f (-2.f, 2.f);  glVertex3f (-8.f, -1.f, -8.f);
 glTexCoord2f (-3.f, 0.f);  glVertex3f (-10.f, -1.f, 0.f);
 glTexCoord2f (-2.f, -2.f); glVertex3f (-8.f, -1.f, 8.f);
}
glEnd ();

glEnable (GL_DEPTH_TEST);
glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
glClear(GL_DEPTH_BUFFER_BIT);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, (GLfloat)
GL_REPLACE);

for (i=0; i<subSize; i++) {
     glBegin (GL_TRIANGLE_STRIP);
      x = -1.f, j=0;
      y  = texture.get_array (i,j);
      if (water_level) if (y < water) y = water;
      glTexCoord2f((x+1.f)*.5f,-((z-1.f)*.5f));glVertex3f(x,y,z);
      y  = textur. Cloud.get_array (I+1,j);
      if (water_level)  if (y < water) y = water;
      glTexCoord2f((x+1.f)*.5f,-((z-1.f+inc)*.5f));
      glVertex3f(x,y, z+inc);
      for (; j<subSize; j++) {
            y2  = textur. Get_array (I,j(1);
            if (water_level)
                  if (y2 < water) y2 = water;
            glTexCoord2f((x+1.f+inc)*.5f,-((z-1.f)*.5f));
            glVertex3f (x+inc, y2, z);
            y1  = textur.get_array (I+1,j+1);
            if (water_level) if (y1 < water) y1 = water;
            glTexCoord2f((x+1.f+inc)*.5f,-((z-1.f+inc)*.5f));
            glVertex3f (x+inc, y1, z+inc);
            x += inc;
      }
      z += inc;
      glEnd ();
}
```

Penyimpanan hasil visualisali kedalam bitmap, dimulai dengan mengambil

BAB V

UJI COBA DAN EVALUASI PERANGKAT LUNAK

BAB V

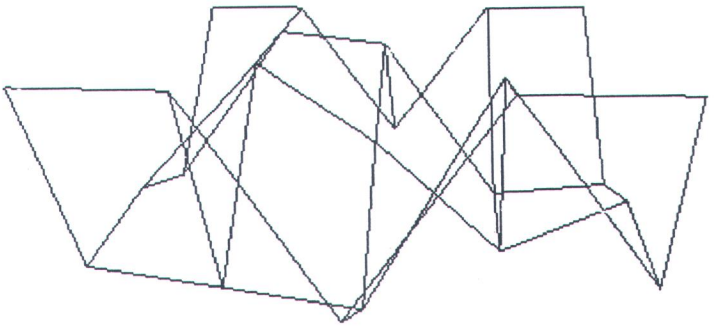
UJI COBA DAN EVALUASI

PERANGKAT LUNAK

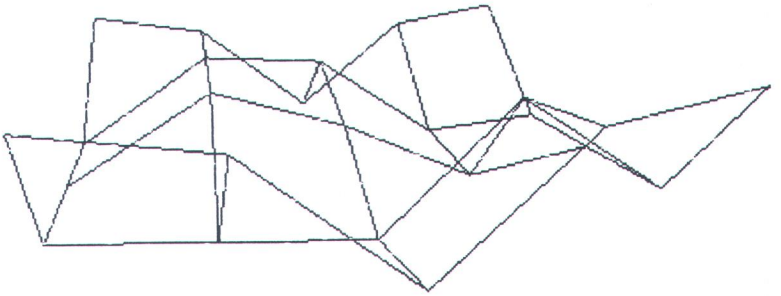
Dalam bab ini dibahas mengenai hasil serangkaian uji coba perangkat lunak dan evaluasi perangkat lunak yang dibuat. Adapun uji coba yang dilakukan adalah dengan memberikan berbagai macam nilai masukan untuk tiap-tiap variabel masukan, disini akan diamati bagaimana keluaran perangkat lunak terhadap masukan tersebut. Sedangkan evaluasi yang dilakukan adalah banyaknya jumlah waktu yang diperlukan oleh perangkat lunak untuk menghasilkan gambar yang diinginkan.

Seperti yang telah dijelaskan pada bagian sebelumnya, banyaknya iterasi yang diberikan oleh pemakai akan menentukan banyaknya jumlah titik yang akan dihasilkan dengan rumusan $(2^n+1)^* (2^n+1)$ dengan n adalah banyaknya iterasi yang dilakukan. Sedangkan faktor kekasaran yang juga dimasukan oleh pengguna menunjukkan bagaimana tingkat kekasaran yang dihasilkan pada permukaan tersebut, semakin kecil nilai yang dimasukan maka semakin kasar permukaan

gambar yang dihasilkan dengan H = 0.7. Hal ini menunjukan semakin rendah nilai H yang diberikan akan menghasilkan permukaan yang semakin kasar.

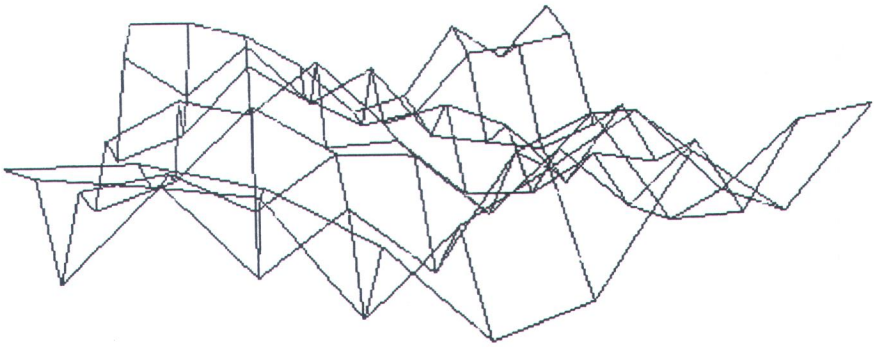


Gambar 5.1 Permukaan yang dihasilkan dengan H=0.2



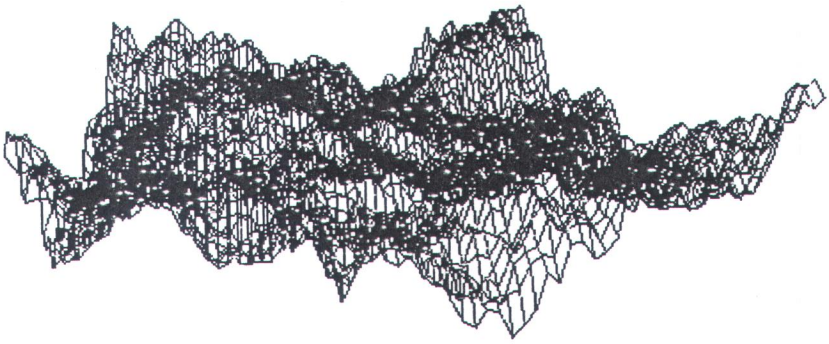
Gambar 5.2 Permukaan yang dihasilkan dengan H=0.7

Kedua gambar diatas (gambar 5.1 dan gambar 5.2) dihasilkan dengan jumlah iterasi sama dengan 2, jumlah titik yang dihasilkan berjumlah 25 titik. Sedangkan gambar 5.3 menunjukan jumlah iterasi sama dengan 3 dan niilai H =



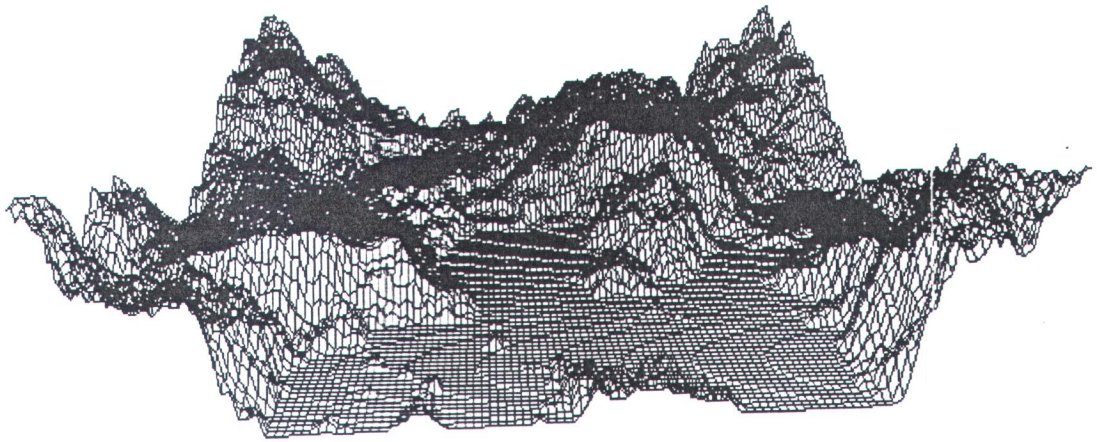
Gambar 5.3 Permukaan yang dihasilkan dengan H=0.7 dan iterasi=3.

Sedangkan gambar yang dihasilkan dengan iterasi=6 dapat dilihat pada gambar 5.4.



Gambar 5.4 Permukaan yang dihasilkan dengan H=0.7 dan iterasi=6.

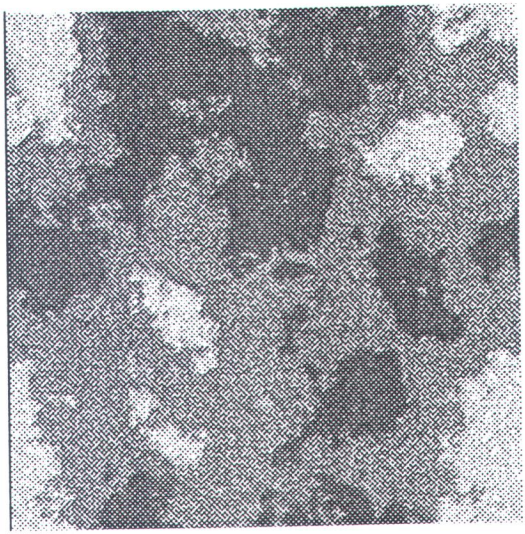
Perangakat lunak mampu menghasilkan permukaan yang mempunyai batas ketinggian air. Dalam gambar 5.5 diperlihatkan suatu permukaan yang dihasilkan dengan batas ketinggian permukaan air 1/5 dari dasar permukaan.



Gambar 5.5 Permukaan dengan ketinggian air.

Hasil pewarnaan yang diterapkan pada permukaan dapat dilihat pada gambar 5.5 dimana banyaknya iterasi yang dilakukan adalah 8, jumlah titik yang pada permukaan tersebut sama dengan 66049 titik. Sedangkan tingkat kekasaran adalah 0.7 dan tanpa adanya unsur ketinggian air. Dan jumlah warna yang diterapkan sebanyak tiga macam warna, ketinggian data permukaan dibagi menjadi tiga bagian dari dasar sampai sepertiga ke atas diberi nilai warna dengan warna pertama masukan pemakai, dan sepertiga tengahnya diberi warna kedua masukan pemakai, sedangkan sepertiga ke atas diberi warna ketiga.

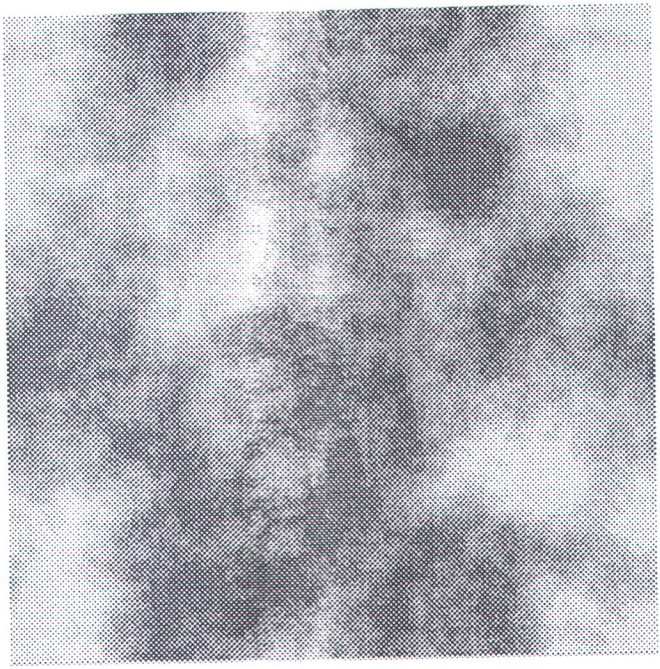
Dari hasil pewarnaan tersebut (gambar 5.6), didapatkan bentuk teksturnya. Yang didapat dengan cara melihat hasil pewarnaan tersebut dari atas. seperti pada gambar 5.7.



Gambar 5.7 Tekstur permukaan yang dihasilkan dari proses pewarr aan.

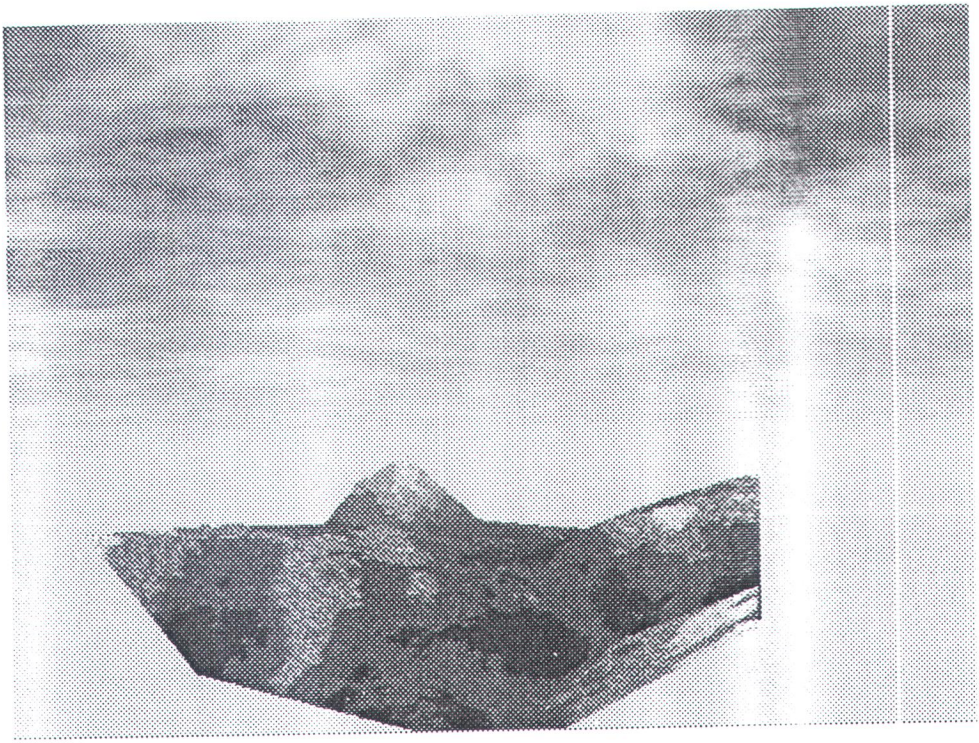
Dari tekstur yang didapat, dapat ditempatkan kepada pola permukaan yang kita inginkan seperti pada gambar 5.8, dimana permukaan yang dilekatkan oleh tekstur, dihasilkan dari H=0.7 dan banyaknya titik yang didapat sebanyak 25 titik, seperti yang terlihat pada gambar 5.2. dan tekstur permukaan yang dipakai adalah tekstur pada gambar 5.7.

Perangkat lunak yang dikembangkan dijuga menghasilkan bentuk awan. Gambar 5.9 memperlihatkan obyek awan yang dibangkitkan dengan nilai H=0.6 dan jumlah iterasi 8.



Gambar 5.9 Awan yang dibentuk dengan metode fraktal

Dari tiap-tiap gambar seperti gambar rangka, tekstur permukaan, dan awan dapat dibuat sebuah gambar yang menjadi satu kesatuan dengan mengabungkan menjadi satu seperti pada gambar 5.10. Gambar 5.10 dihasilkan dengan nilai H sama dengan 0.7 iterasi untuk permukaan sama dengan 2, iterasi tekstur



Gambar 5.10 Hasil pewarnaan dan penteksturan

Berikut ini ditampilkan tabel 5.1 perbandingan waktu yang dibutuhkan oleh perangkat lunak guna menghasilkan gambar rangka (*wire frame*). Perhitungan waktu dilakukan dalam detik. Perbandingan yang dilakukan adalah jumlah iterasi dan bentuk kekasaran permukaan yang dibuat berdasarkan masukan nilai H.

Table 5.1 Waktu proses untuk nilai H dan jumlah Iterasi tertentu

pembuatan gambar rangka

permukaan yang dihasilkan semakin kasar sehingga jarak antara satu titik ke titik semakin besar, yang menyebabkan waktu pembuatan gambar semakin lama.

Tabel 5.2 Waktu proses untuk nilai H dan jumlah Iterasi tertentu

pewarnaan permukaan

	2	3	4	5	6	7	8	9	10
H=0.7	0.0257	0.0317	0.0451	0.1046	0.2918	0.8840	3.2110	11.215	45.793
H=0.5	0.0308	0.0560	0.1123	0.1503	0.3096	0.9575	3.3665	12.956	49.624
H=0.3	0.1889	0.0632	0.0664	0.1409	0.3966	1.2426	4.2513	15.920	61.418
H=0.1	0.0329	0.0513	0.0881	0.2118	0.6242	2.0343	7.0249	25.635	102.60

BAB V

KESIMPULAN DAN SARA

BAB VI

KESIMPULAN DAN SARAN

6.1. KESIMPULAN

Dalam tugas akhir ini terdapat beberapa kesimpulan yang dapat ditarik dari serangkaian uji coba terhadap perangkat lunak :

1. *Terrain map* yang dihasilkan dengan menggunakan metode fraktal, dalam hal ini menggunakan metode *midpoint-displacement,* secara visual cukup representatif untuk permukaan alam.
2. Tingkat kekasaran dari terrain yang dibentuk ditentukan oleh nilai H. Semakin besar nilai tersebut maka permukaan yang dihasilkan akan semakin halus, dan kebalikannya semakin rendah nilai H maka permukaan tampak lebih kasar.

6.2. SARAN

Perangkat lunak yang dibuat masih memiliki kekurangan dan memerlukan tambahan kelengkapan. Berikut ini beberapa saran yang dapat digunakan sebagai bahan pengembangan perangkat lunak ini lebih lanjut :

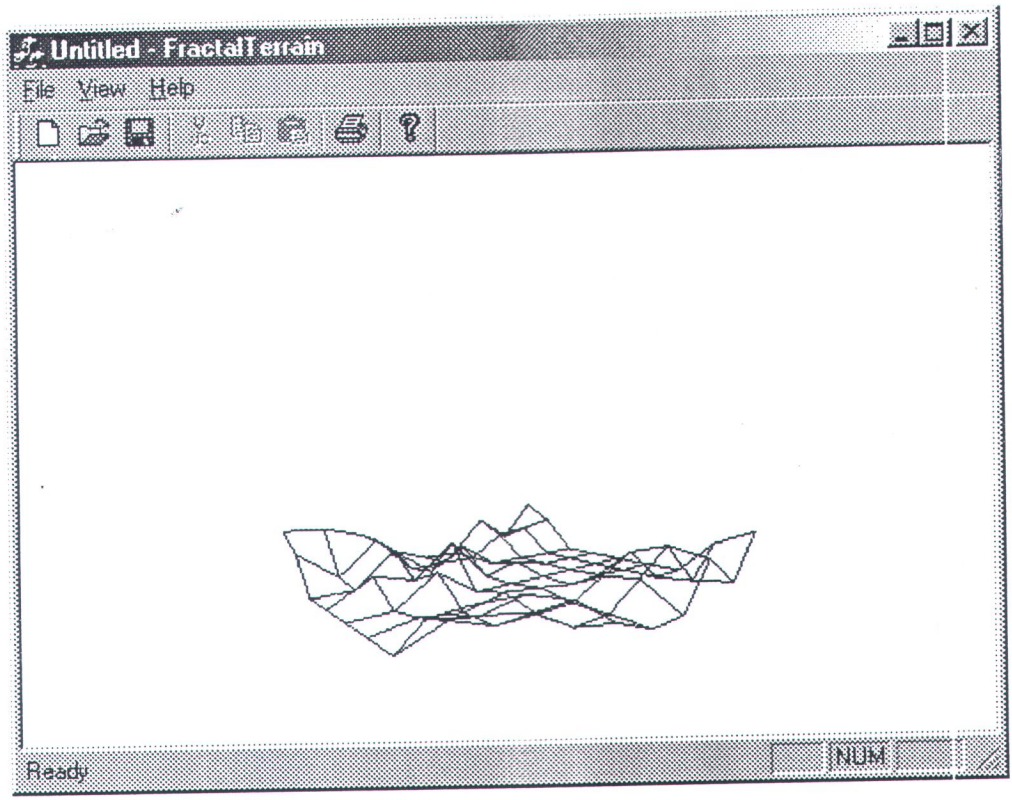
3. Pembentukan permukaan dengan menggunakan bentuk dasar yang telah tentu, tanpa menggunakan nilai random.
4. Pada proses pewarnaan yang dilakukan dapat digunakan metode *Phong shading* untuk menghasilkan gambar yang lebih bagus.

LAMPIRAN

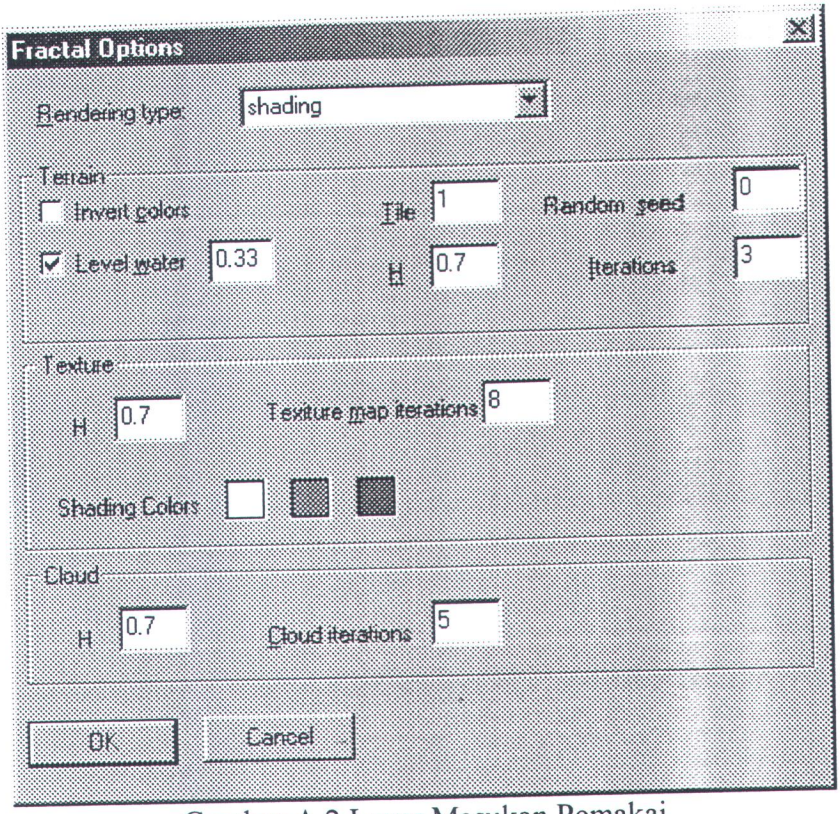
LAMPIRAN A

PETUNJUK PENGGUNAAN PERANGKAT LUNAK

Seperti telah dijelaskan pada bagian sebelumnya perangkat lunak yang dibuat gan menggunakan Microsoft Visual C++ dengan sistem operasi windows. Gambar memperlihatkan tampilan perangkat lunak pembagkit terrain map dengan ggunakan nilai kekasaran H = 0.7 dan jumlah iterasi sebanyak 3.



Gambar A.1 Tampilan Perangkat Lunak



Gambar A.2 Layar Masukan Pemakai

Struktur menu dari perangkat lunak dapat dilihat pada tabel A.1, dan juga gsi dari tiap item-item menu tersebut.

Tabel A.1 Struktur Menu

up Item	Menu Item	Fungsi
	New	Membuat terrain baru
	Save	Menyimpan gambar terrain kedalam format bitmap
	Save as ...	Menyimpan gambar
	Print	Mencetak gambar
	Print Preview	Melihat hasil cetak pada layar monitor

		tekstur
	Render	Menampilkan gambar dari gabungan
	Options	Menampilkan layar masukan pemakai
	About	Menampilkan layar informasi perangkat lunak