BAB IV

IMPLEMENTASI DAN EVALUASI

4.1. Kebutuhan Sistem

Agar dapat mengimplementasikan dan menjalankan framework rendering engine ini dibutuhkan perangkat keras dan perangkat lunak dengan spesifikasi tertentu. Adapun kebutuhan perangkat keras dan perangkat lunak untuk sistem ini adalah sebagai berikut:

4.1.1. Kebutuhan Perangkat Keras

Kebutuhan perangkat keras yang akan digunakan dalam pembuatan dan implementasi dari framework rendering engine ini adalah sebagai berikut:

1. *Processor* Intel Core 2 Duo atau lebih (mendukung MMX dan SSE)
2. Memory 2 GB atau lebih.
3. Harddisk 80 GB atau lebih.
4. Monitor VGA dengan resolusi 800 x 600 keatas.
5. VGA 3D Nvidia atau AMD ATI dengan memory 128 MB keatas.
6. Perlengkapan tambahan seperti *mouse* dan *keyboard*.

4.1.2. Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak yang digunakan dalam pembuatan framework rendering engine ini adalah sebagai berikut:

1. Sistem operasi menggunakan Microsoft Windows 7 *Professional*.
2. Microsoft Visual Studio 2008 Professional Edition.
3. Microsoft DirectX SDK 9.0c (*December* 2006).

STIKOM SURABAYA

4. Rational Rose Enterprise Edition 2003.
5. Microsoft Visio 2010 Professional.

Sedangkan untuk implementasi framework rendering engine ini kebutuhan minimum dari perangkat lunaknya adalah sebagai berikut:

1. Sistem operasi Microsoft Windows XP.
2. Microsoft Visual Studio 2005.
3. Microsoft DirectX SDK 9.0c (December 2006).

4.2. Pembuatan Framework Rendering Engine

Berdasarkan hasil analisa dan rancangan sistem yang dibuat, maka langkah selanjutnya adalah tahapan pembuatan rancang bangun sistem menggunakan bahasa pemrograman C++ dan DirectX SDK (Software Development Kit).

Bahasa C++ dipilih karena mampu menangani pemrograman secara luas baik dari tingkat rendah hingga tingkat tinggi. Selain itu C++ juga bahasa standar yang digunakan untuk pengembangan yang menggunakan DirectX SDK. Dengan demikian untuk pembuatan kode program, perangkat lunak yang digunakan adalah menggunakan Microsoft Visual Studio 2006 Professional Edition.

Pembuatan Tumoz Framework Rendering Engine ini di buat menjadi empat *project* (subsistem) dalam satu *solution* dengan nama TumozEngine. Project tersebut di bagi berdasarkan fungsi dan tugas masing-masing seperti yang dijelaskan pada analisa dan desain sistem, yaitu terdiri dari TumozRenderer, TumozD3D, Tumo3D, dan TumozGeneral. Dan hasil akhir kompilasi dari project tersebut akan menjadi libary dalam bentuk file lib dan dll agar dapat diimplementasikan.

STIKOM SURABAYA

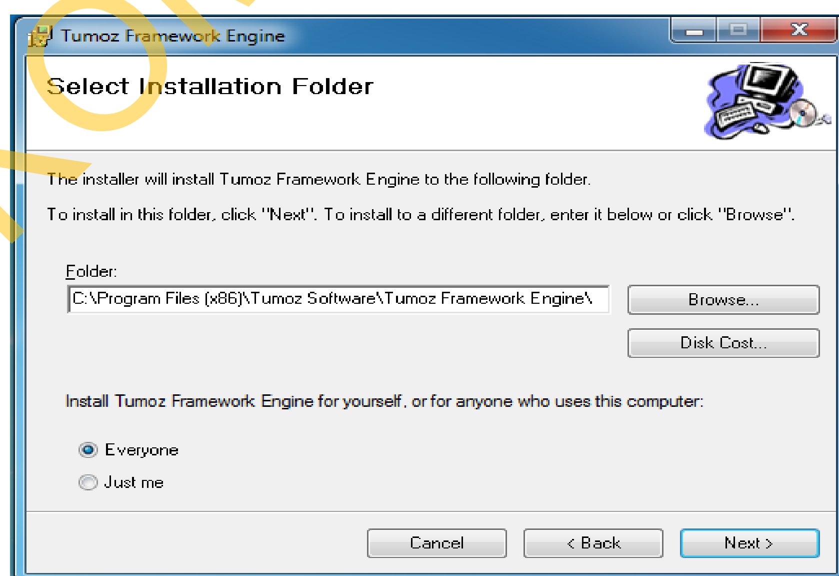
4.3. Implementasi Framework Rendering Engine

Tahapan selanjutnya adalah melakukan implementasi framework rendering engine setelah kebutuhan perangkat keras dan perangkat lunak telah terpenuhi yaitu dengan penjelasan sebagai berikut.

4.3.1. Instalasi Framework Rendering Engine

Untuk dapat menggunakan framework rendering engine ini maka harus dilakukan instalasi terlebih dahulu. Instalasi pada framework ini merupakan faktor terpenting karena akan menjadi acuan (referensi) utama bagi pengembang untuk dapat memanfaatkan semua fitur dan fungsi yang ada di dalam framework tersebut.

File *setup.exe* dijalankan untuk memulai instalasi dan setelah itu akan muncul *form* yang berisi halaman awal. Pada halaman awal tersebut klik tombol *next* untuk menuju ke halaman pemilihan folder instalasi yang diinginkan seperti yang terlihat pada gambar 4.1. Setelah itu klik tombol next untuk menuju halaman konfirmasi dan klik tombol next sekali lagi untuk memulai proses instalasi.



Gambar 4.1. Halaman Pemilihan Folder Instalasi

STIKOM SURABAYA

Jika instalasi framework rendering engine tersebut berjalan lancar maka akan menuju ke form akhir yang menunjukkan bahwa instalasi telah selesai dan kemudian klik tombol *close* untuk menutup program instalasi tersebut. Dengan demikian maka framework rendering engine siap untuk digunakan.

Folder hasil instalasi framework rendering engine tersebut berisi file-file yang dapat digunakan untuk implementasi pengembangan aplikasi berbasis tiga dimensi, yaitu terdiri dari file-file header, file-file library (.lib dan .dll), file-file gambar untuk logo dan *icon* engine, dan terakhir adalah file-file dokumentasi untuk keterangan penggunaan framework rendering engine.

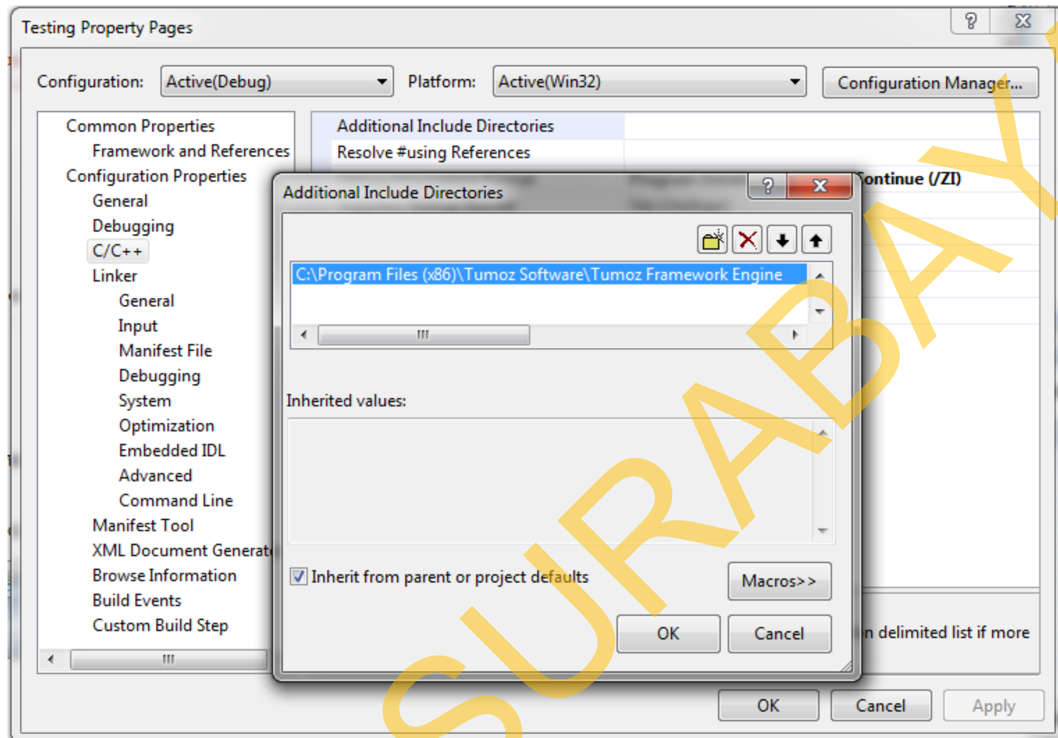
4.3.2. Pembuatan Referensi Framework Rendering Engine

Untuk membangun aplikasi berbasis tiga dimensi yang menggunakan framework rendering engine ini, maka langkah awal yang harus dilakukan adalah membuat referensi atau acuan ke file-file header dan libary dari framework rendering engine. Hal ini ditujukan agar program aplikasi yang dikembangkan dapat mengakses fungsionalitas dari framework tersebut.

Untuk membuat acuan file header pada project pengembangan aplikasi ada 2 macam cara, yaitu dengan menyalin (*copy*) file-file header yang dibutuhkan ke dalam folder project atau menggunakan halaman *property* yang disediakan oleh IDE (*Integrated Development Environment*). Contoh cara mengakses halaman property pada IDE Visual Studio 2008 yaitu dengan cara menekan tombol Alt+F7 pada *keyboard* lalu pilih *Configuration Properties* lalu pilih *C/C++* dan terakhir pilih *Additional Include Directories* agar dapat memasukkan lokasi folder file header yang diinginkan seperti yang dicontohkan pada gambar 4.2. Setelah itu file header sebagai *forward declaration* dari kelas-kelas, fungsi-fungsi (*subroutine*),

STIKOM SURABAYA

variabel-variabel, dan *identifier* lainnya dapat diakses pada saat pengkodean menggunakan bahasa C++ setelah menulis perintah kata kunci (*keyword*) *#include directive* beserta nama filenya.

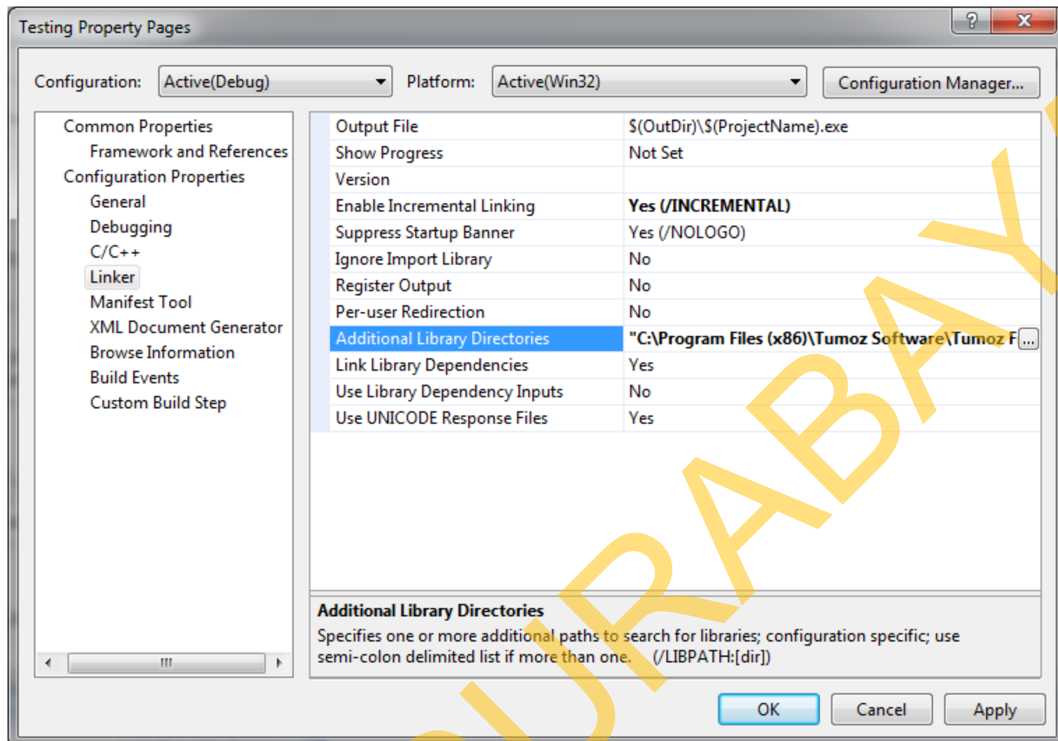


Gambar 4.2. Halaman Additional Include Directories

Selain membuat acuan atau referensi ke file header, file library juga perlu dibuatkan acuannya untuk proses *linking* pada saat kompilasi. Caranya hampir sama dengan membuat referensi atau acuan pada file header yaitu pertama dengan cara menyalin atau mengkopi file-file library (.lib dan .dll) yang dibutuhkan ke dalam folder project, atau kedua dengan menggunakan halaman property yang disediakan oleh IDE. Apabila menggunakan IDE Visual Studio 20008, maka cara mengaksesnya adalah dengan menekan tombol Alt+F7 pada keyboard lalu pilih Configuration Properties lalu pilih Linker dan terakhir pilih *Additional Library*

STIKOM SURABAYA

Directories untuk memasukkan lokasi folder file library yang diinginkan seperti pada gambar di bawah ini.



Gambar 4.3. Halaman Additional Library Directories

Untuk informasi tambahan bahwa file TumozD3D.dll harus tetap berada di dalam folder project pengembangan aplikasi, agar tidak mengalami error pada saat proses kompilasi. Selain itu file gambar tumoz.bmp juga harus dimasukkan atau disalin ke dalam folder project pengembangan aplikasi, agar gambar logo engine dapat ditampilkan.

4.3.3. Insialisasi Framework Rendering Engine

Inisialiasi dapat dilakukan setelah handle *instance* windows telah di buat terlebih dahulu (pelajari pemrograman windows dasar C++ menggunakan API *Win32*). Setelah itu baru fungsi *Tumoz3DInitCPU()* di panggil untuk mengaktifkan CPU matematika cepat SIMD jika tersedia.

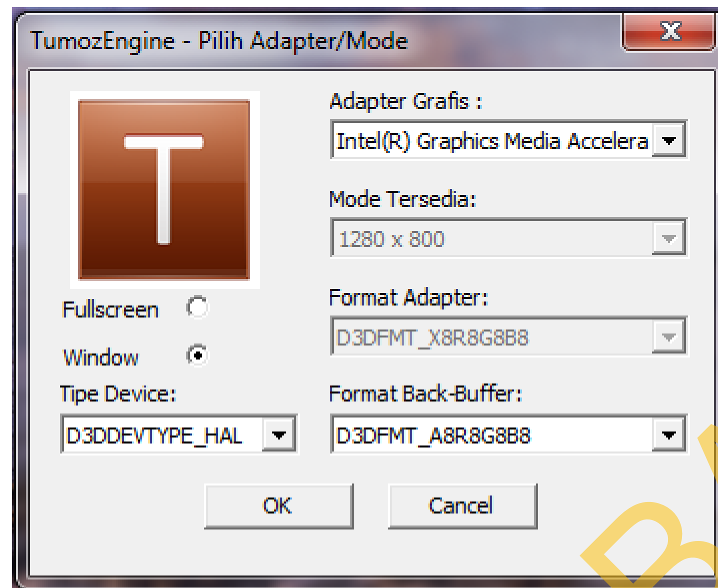
STIKOM SURABAYA

Proses insialisasi membutuhkan dua pointer yaitu dari obyek kelas TumozRenderer dan TumozRenderDevice agar engine dapat berjalan. Obyek TumozRenderer di buat berdasarkan handle instance windows sebagai parameter dari konstruktor. Lalu kemudian perangkat rendering di buat dengan cara memanggil fungsi CreateDevice() dari obyek TumozRenderer dengan API_DIRECT3D sebagai parameternya yang menandakan penggunaan API Direct3D. Perangkat rendering tersebut dapat digunakan oleh obyek TumozRenderDevice dengan cara memangil fungsi *GetDevice()* dari obyek TumozRenderer dan memasukkan hasilnya ke dalam pointer obyek TumozRenderDevice, sehingga dengan demikian engine siap untuk diinsialisasi.

Inisialisasi pada obyek TumozRenderDevice terdiri dari 2 macam fungsi yaitu Init() dan InitWindowed(). Fungsi inisialisasi yang paling umum digunakan adalah fungsi Init() dengan parameter handle windows utama, array handle child windows untuk beberapa tampilan jendela, jumlah child windows, jumlah minimum depth bit, jumlah minimum stencil bit, dan terakhir nilai *boolean* untuk mengaktifkan log. Apabila fungsi tersebut dieksekusi maka akan tampil form untuk memilih adapter grafis, resolusi layar, format adapter, format back-buffer, dan sebagainya seperti pada gambar 4.4.

Sedangkan fungsi inisialisasi InitWindowed() juga memiliki beberapa parameter yaitu handle windows utama, array handle child windows, jumlah child windows, dan nilai boolean untuk mengaktifkan log. Berbeda dari fungsi sebelumnya fungsi ini tidak menampilkan form untuk memilih adapter, format tampilan layar, dan sebagainya, akan tetapi format tersebut harus dideklarasikan sendiri pada saat menulis kode program aplikasi.

STIKOM SURABAYA



Gambar 4.4. Form Memilih Adapter dan Format

Setelah mengeksekusi salah satu dari kedua fungsi inisialisasi tersebut baik menggunakan fungsi Init() ataupun menggunakan fungsi InitWindowed(), maka fungsi-fungsi tersebut akan menyimpan format pilihan adapter, tampilan layar, dan beberapa format lainnya sesuai dengan keinginan user. Dengan demikian proses inisialisasi telah berakhir dan windows utama siap untuk menampilkan jalannya proses rendering.

4.3.4. Siklus Rendering

Siklus rendering terjadi di mana engine ini melakukan penggambaran suatu scene secara berulang-ulang sehingga terjadi animasi yang berubah secara dinamis dan interaktif, yang berdasarkan kondisi-kondisi tertentu yang telah ditentukan sebelumnya oleh pengembang aplikasi. Sebelum perulangan dalam proses rendering terjadi, pihak pengembang harus menentukan terlebih dahulu kondisi-kondisi atau pengaturan scene yang diinginkan, contohnya yaitu: posisi arah pandang, warna *background*, viewport stage ,dan sebagainya. Pengaturan

STIKOM SURABAYA

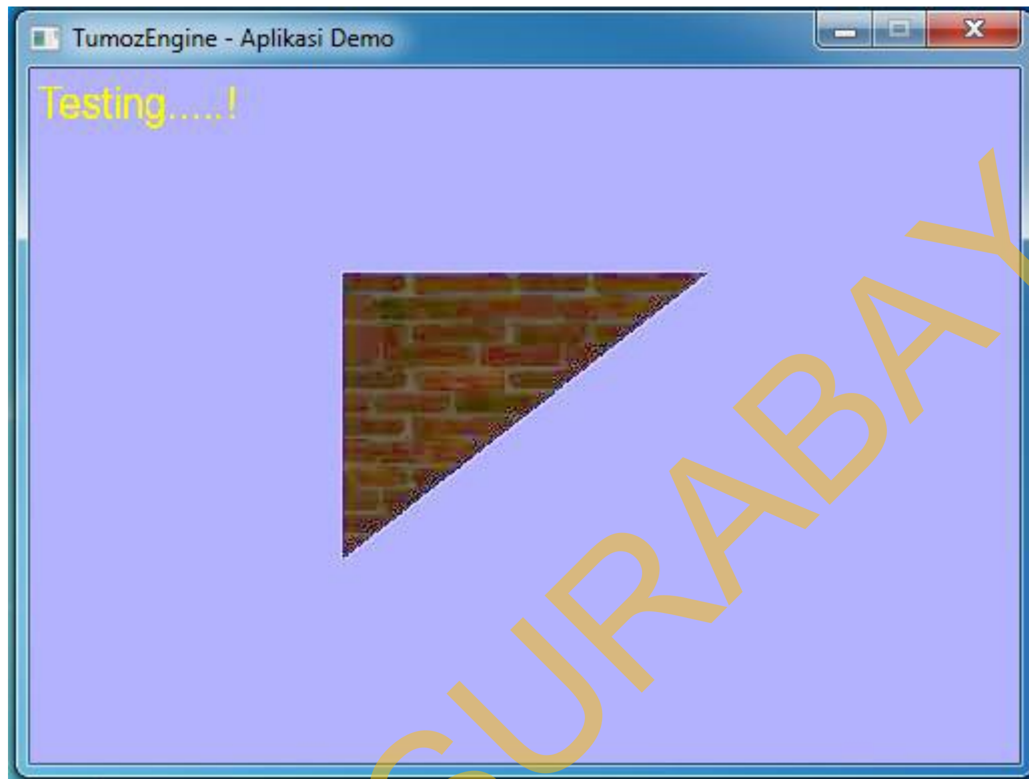
scene atau kondisi tersebut merupakan fungsi-fungsi bagian dari perangkat rendering, yakni obyek dari kelas TumozRenderDevice, di mana biasanya dilakukan setelah proses inisialisasi selesai dan pada saat perangkat rendering telah aktif atau siap digunakan.

Untuk pengaturan awal setelah inisialisasi, pihak pengembang biasanya menentukan target windows mana yang akan melakukan proses rendering. Karena pada saat inisialisasi, framework rendering engine diijinkan untuk menggunakan beberapa windows secara bersamaan. Fungsi yang bertugas untuk itu adalah fungsi UseWindow(), dengan nilai indeks dari array handle child windows sebagai parameternya sesuai dengan jumlah windows yang ditentukan pada saat inisialisasi. Pengaturan selanjutnya adalah mengatur viewport stage. Tujuan dari pengaturan ini adalah untuk mengkalkulasi pada masing-masing stage, matriks proyeksi perspektif dan orthogonalnya untuk memproyeksikan scene tiga dimensi menjadi gambar dua dimensi di dalam daerah tampilan layar. Hal tersebut dapat dilakukan dengan memanggil fungsi InitStage(), dengan parameternya antara lain: nilai field of view (memberikan efek *zoom* pada scene), dimensi viewport (jika diberikan nilai NULL, maka secara otomatis menghitung dimensi layar), dan indeks stage yang akan diatur (maksimal 4 stage).

Untuk mengatur posisi arah pandang pada scene (posisi kamera), framework rendering engine menggunakan perhitungan transformasi matriks untuk operasinya yaitu dengan menggunakan fungsi *SetView3D()*. Parameter dari fungsi tersebut terdiri dari vektor kanan kamera, vektor atas kamera, vektor arah kamera, dan terakhir vektor posisi kamera. Sedangkan untuk mengatur kondisi warna background atau warna untuk membersihkan layar dari scene, framework

STIKOM SURABAYA

rendering engine ini menggunakan fungsi *SetClearColor()* dengan 4 nilai parameter warna yaitu merah, hijau, biru, dan alpha.



Gambar 4.5. Siklus rendering sederhana

Setelah semua kondisi dari scene telah diatur, maka siklus perulangan dalam proses rendering dapat dimulai. Pada pemrograman windows, siklus ini biasanya dilakukan pada saat proses menerjemahkan *message* dari fungsi *Callback Windows Procedure* (pelajari pemrograman windows dasar C++ menggunakan API Win32). Untuk memulai proses rendering, obyek perangkat rendering dari kelas TumozRenderDevice memanggil fungsi BeginRendering() dengan 3 parameter berupa nilai boolean, yaitu nilai untuk membersihkan pixel buffer, depth buffer, dan terakhir stencil buffer. Setelah operasi tersebut pihak pengembang dapat memanggil beragam fungsi-fungsi untuk menggambar berbagai macam obyek tiga dimensi beserta efek-efek animasinya, dan

STIKOM SURABAYA

perhitungan perubahan transformasinya. Setelah itu siklus ini diakhiri dengan memanggil fungsi EndRendering() dari obyek perangkat rendering kelas TumozRenderDevice, untuk mengakhiri proses siklus rendering dan menampilkan scene ke front buffer (ke dalam layar monitor).

4.3.5. Rendering Obyek Tiga Dimensi

Untuk menggambar suatu obyek tiga dimensi pada framework rendering engine, dibutuhkan dua buah obyek manager untuk mengelola proses rendering dari obyek tiga dimensi tersebut. Obyek manager yang pertama bertugas untuk membentuk wujud dari obyek tiga dimensi sesuai dengan keinginan dari pihak pengembang, dan ini merupakan tugas dari obyek kelas TumozVertexCacheManager. Sedangkan obyek manager yang lain bertugas untuk memberikan warna dan tekstur pada lapisan terluar atau kulit dari obyek tiga dimensi tersebut, yang merupakan tugas dari obyek kelas TumozSkinManager.

Sebelum membentuk obyek tiga dimensi dengan menggunakan verteks, warna dan tekstur dari obyek tersebut harus didefinisikan terlebih dahulu. Untuk membuat skin dan warna dasarnya, adalah dengan memanggil fungsi AddSkin() yang merupakan bagian dari kelas TumozSkinManager, yang di panggil melalui fungsi *GetSkinManager()* dari obyek TumozRenderDevice. Fungsi AddSkin() dipanggil berserta parameternya yang terdiri dari 4 obyek kelas *structure* TUMOZCOLOR untuk masing-masing warna ambient, diffuse, *emissive*, dan *specular*, lalu parameter lainnya adalah nilai integer untuk specular *power,* dan nilai integer untuk menyimpan id skin sebagai output. Apabila obyek tiga dimensi tersebut membutuhkan tekstur, maka fungsi AddTexture() dipanggil dengan parameter yang terdiri dari id skin yang telah di buat sebelumnya, nama

STIKOM SURABAYA

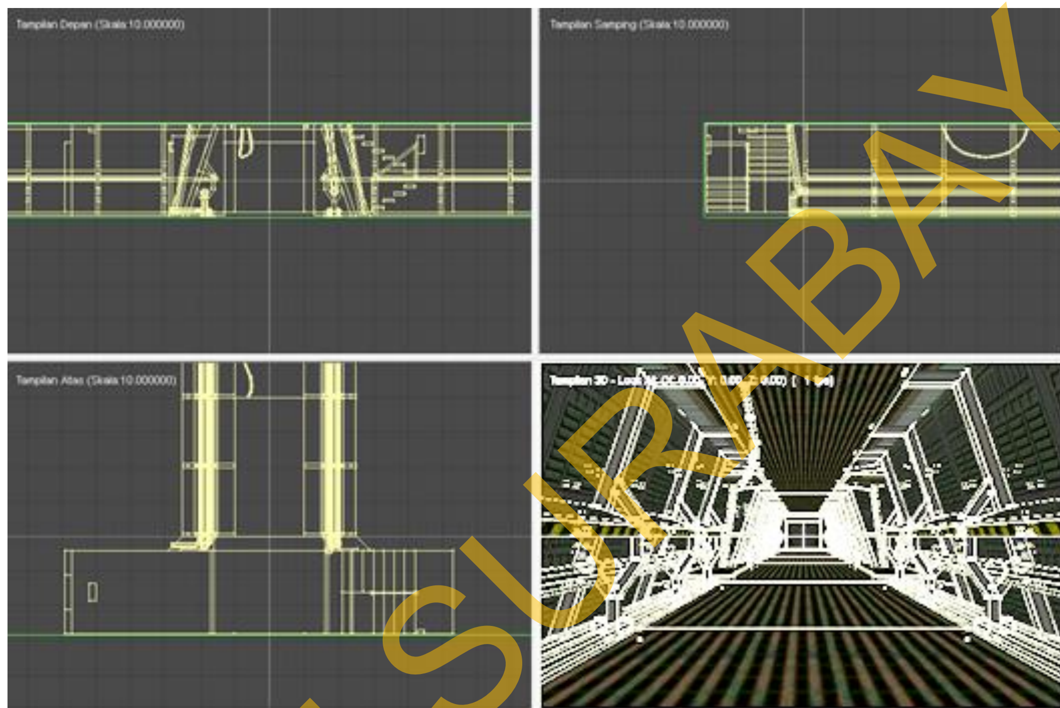
file tekstur (dalam format BMP), nilai boolean alpha (untuk menggunakan alpha blending), nilai float transparansi keseluruhan, array warna color *keys* menggunakan kelas structure TUMOZCOLOR, dan terakhir jumlah array warna color keys.

Langkah selanjutnya adalah verteks dari obyek tiga dimensi tersebut di buat dengan cara mendefinisikan array dari salah satu berbagai macam jenis dari kelas *stucture* verteks yang sudah disediakan oleh framework rendering engine, yaitu terdiri dari kelas structure VERTEX, LVERTEX, TVERTEX, PVERTEX, CVERTEX, dan VERTEX3T. Kemudian property atau membernya diisi dengan data-data yang diperlukan untuk membentuk suatu obyek tiga dimensi, seperti koordinat posisi verteks, koordinat tekstur, vektor normal dan sebagainya. Jika menggunakan indeks maka harus ditentukan juga, yaitu dengan cara dengan mendefinisikan array dari tipe data WORD. Apabila verteks telah diatur maka langkah selanjutnya adalah memanggil fungsi CreateStaticBuffer() bagian dari obyek kelas TumozVertexCacheManager, yang di panggil melalui obyek TumozRenderDevice dengan memakai fungsi *GetVertexManager()*. Fungsi CreateStaticBuffer() dipanggil berserta parameternya, yaitu jenis verteks dari member *enumeration* TUMOZVERTEXID, id skin yang akan digunakan, jumlah array verteks, jumlah array indeks, data array verteks, data array indeks, dan id verteks manager sebagai output.

Setelah obyek tiga dimensi telah didefinisikan baik verteks maupun skinnya, maka obyek tiga dimensi tersebut akan di gambar (di render) dalam ruang lingkup (perulangan) siklus rendering dengan memanggil fungsi Render() yang merupakan bagian dari obyek kelas TumozVertexCacheManager, yang di

STIKOM SURABAYA

panggil melalui obyek TumozRenderDevice dengan memakai fungsi GetVertexManager(). Parameter dari fungsi Render() tersebut ada bermacam-macam tetapi yang paling umun digunakan hanya memakai satu parameter saja, yaitu id verteks manager yang telah di buat sebelumnya.



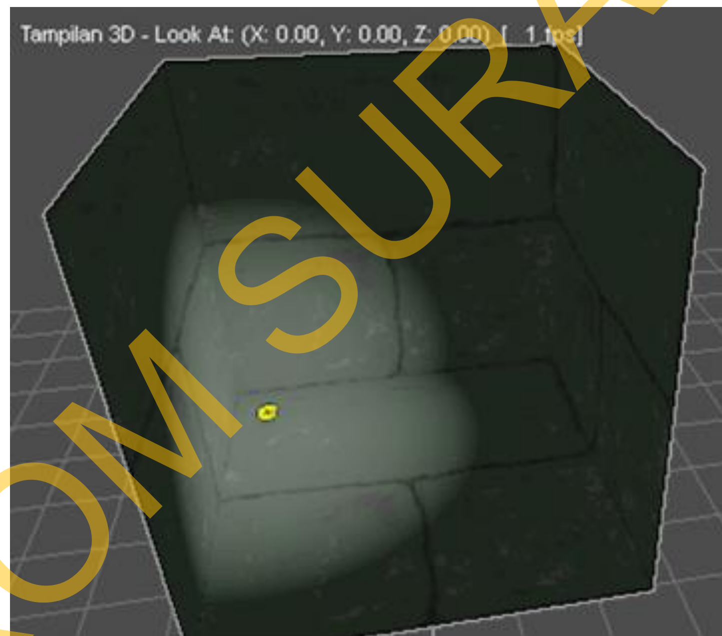
Gambar 4.6. Rendering Obyek Tiga Dimensi

4.3.6. Transformasi dan Pencahayaan

Menggambar obyek tiga dimensi saja tidak akan cukup untuk membuat rendering bergerak secara dinamis dan memberikan efek agar terkesan lebih nyata. Oleh karena itu proses rendering harus diberikan beberapa teknik pemrograman tiga dimensi dasar, yaitu tranformasi dan pencahayaan. Teknik-teknik tersebut dapat dilakukan pada saat pengaturan kondisi setelah inisialisasi untuk efek statis, dan atau pada saat terjadinya proses siklus rendering untuk memberikan efek dinamis.

STIKOM SURABAYA

Teknik transformasi merupakan teknik perubahan bentuk dan pergerakan dari suatu obyek tiga dimensi, yang menggunakan perhitungan matematika seperti matriks, quaternion, dan sebagainya. Framework rendering engine memfasilitasi teknik tersebut dengan menyediakan kelas-kelas matematika tiga dimensi seperti TumozVector, TumozMatrix dan sebagainya. Dan obyek perangkat rendering dari kelas TumozRenderDevice akan mengeksekusi transformasi tersebut secara keseluruhan menggunakan fungsi SetWorldTransform() dengan parameter pointer dari obyek kelas TumozMatrix, yang merupakan gabungan transformasi dari semua obyek yang terjadi di dalam scene.



Gambar 4.7. Efek Pencahayaan Pada Scene

STIKOM SURABAYA

Teknik pencahayaan ditujukan untuk memberikan kesan nyata (efek realisme) yang tampak pada scene, seperti efek cahaya sinar lampu, sorotan cahaya matahari, dan bermacam-macam efek pencahayaan lainnya. Obyek TumozRenderDevice menyediakan dua fungsi untuk teknik pencahayaan ini, yaitu fungsi SetAmbientLight(), dan *SetLight()*. Operasi atau fungsi

SetAmbientLight() memberikan efek pencahayaan yang menerangi secara merata di seluruh scene (contohnya cahaya sinar matahari). Parameternya terdiri dari 3 nilai float yang mewakili besarnya nilai warna merah, hijau, dan biru. Sedangkan fungsi SetLight() memberikan beragam efek pencahayaan untuk stage tertentu. Parameter fungsi tersebut terdiri dari kelas structure TUMOZLIGHT (property atau membernya terdiri dari tipe sumber pencahayaan, warna diffuse, warna specular, warna ambient, jarak cahaya, sudut theta, sudut phi, atenuasi0 , dan atenuasi1), dan indeks stage yang akan di beri pencahayaan.

4.3.7. Implementasi Shader

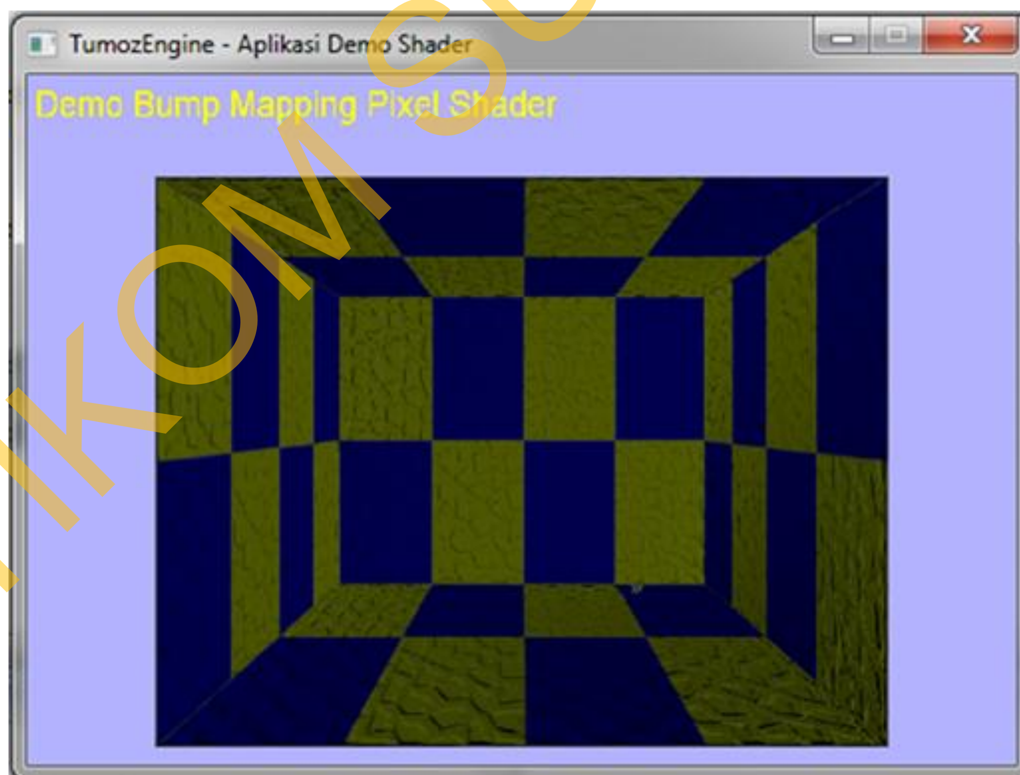
Sebelumnya kita membuat obyek tiga dimensi yaitu berupa gabungan verteks berdasarkan kelas-kelas structure yang telah disediakan oleh framework rendering engine sebelumnya, dan juga memproses beberapa teknik seperti pencahayaan dan transformasi dari fungsi-fungsi yang juga telah disediakan oleh framework rendering engine. Akan tetapi ada kalanya pihak pengembang harus menggunakan jenis verteks yang harus dikustomisasi dan di olah secara khusus sesuai dengan kebutuhannya. Dan selain itu pihak pengembang mungkin juga menginginkan untuk memproses teknik efek-efek khusus tingkat lanjut (advanced) yang tidak disediakan oleh fungsi-fungsi pada framework rendering engine sebelumnya, contohnya seperti *bumpmapping*, *grayscaling*, *multitexturing,* dan sebagainya. Untuk mengatasi masalah tersebut framework rendering telah menyediakan fasilitas pemrograman shader yang dapat berjalan pada GPU dengan menggunakan bahasa pemrograman HLSL (*High* Level Shader Language).

Ada 2 macam jenis pemrograman shader yang disediakan oleh framework rendering engine yaitu Vertex Shader, dan Pixel Shader. Vertex shader

STIKOM SURABAYA

berjalan ketika masing-masing verteks diserahkan ke GPU untuk diolah, yang mana ditujukan untuk dapat memanipulasi property seperti posisi, warna, dan koordinat tekstur. Sedangkan Pixel shader dieksekusi oleh GPU pada saat proses rasterization untuk masing-masing pixel, yang ditujukan untuk melakukan komputasi untuk warna dan atribut-atribut lainnya pada setiap pixel.

Untuk dapat menggunakan shader pada saat proses rendering, shader tersebut harus dibuat terlebih dahulu dengan memanggil fungsi CreateVShader() dan CreatePShader(), untuk masing-masing vertex dan pixel shader pada obyek perangkat rendering TumozRenderDevice. Kedua fungsi tersebut memiliki parameter yang sama, yaitu terdiri dari nama file atau data *stream* shader yang akan dijalankan, nilai boolean penggunaan file, nilai boolean shader telah dikompilasi, dan nilai integer untuk menyimpan id shader sebagai output.



Gambar 4.8. Spesial Efek Bumpmapping Menggunakan Shader

STIKOM SURABAYA

Sebelum mengaktifkan shader tersebut di dalam siklus rendering, pihak pengembang dapat mengatur nilai variabel konstan baru ke dalam proses eksekusi shader dengan menggunakan fungsi *SetShaderConstant()* dari obyek TumozRenderDevice. Parameternya yaitu berupa nilai member dari enumerasi TUMOZSHADERTYPE untuk menentukan jenis shadernya (vertex atau pixel), nilai member dari enumerasi TUMOZDATATYPE untuk menentukan tipe data variabel konstannya (boolean, integer, atau float), nilai indeks register awal, jumlah array data, dan data array nilai variabel konstan.

Jika shader sudah siap untuk dijalankan dalam proses siklus rendering, maka obyek perangkat rendering TumozRenderDevice dapat mengaktifkannya dengan dengan memanggil fungsi ActivateVShader() untuk vertex shader, dengan parameter id vertex shader yang telah di buat sebelumnya, dan nilai member dari enumerasi TUMOZVERTEXID untuk menentukan jenis verteks yang digunakan. Sedangkan untuk mengaktifkan pixel shader obyek TumozRenderDevice memanggil fungsi ActivatePShader() dengan nilai id pixel shader yang telah dibuat sebelumnya sebagai satu-satunya parameter.

4.4. Pengujian Sistem Rendering

Tujuan dari pengujian ini adalah untuk menganalisa seberapa efektif dan efisien penggunaan sistem rendering pada framework rendering engine dengan membandingkan dengan implementasi dari sistem rendering API grafis lainnya, dalam ini adalah API DirectX. Berikut ini adalah adalah penjelasan perbandingan masing-masing tahapan dalam memprogram sebuah aplikasi tiga dimensi sederhana:

STIKOM SURABAYA

4.4.1. Pengujian Perbandingan Inisialisasi Sistem

Pada proses inisialisasi, API DirectX harus mendeklarasikan semua parameter device yang dibutuhkan untuk pembuatan aplikasi 3D sebelum ditampilkan yaitu seperti kemampuan kartu grafis (pemrosesan hardware atau software), pengaturan back buffer, format warna, jumlah buffer, jenis multi sample, stencil, dan sebagainya. Berikut adalah contoh kode inisialisasi menggunakan API DirectX:

```
// INISIALISASI

g_pD3D = Direct3DCreate9(D3D_SDK_VERSION);

if(!g_pD3D) {
        MessageBox(0, "Direct3DCreate9() - GAGAL", 0, 0);
        return false;
}

D3DCAPS9 caps;
g_pD3D->GetDeviceCaps(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, &caps);

int vp = 0;
if( caps.DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT )
        vp = D3DCREATE_HARDWARE_VERTEXPROCESSING;
else
        vp = D3DCREATE_SOFTWARE_VERTEXPROCESSING;

D3DPRESENT_PARAMETERS d3dpp;
d3dpp.BackBufferWidth            = 500;
d3dpp.BackBufferHeight           = 375;
d3dpp.BackBufferFormat           = D3DFMT_A8R8G8B8;
d3dpp.BackBufferCount            = 1;
d3dpp.MultiSampleType            = D3DMULTISAMPLE_NONE;
d3dpp.MultiSampleQuality         = 0;
d3dpp.SwapEffect                 = D3DSWAPEFFECT_DISCARD;
d3dpp.hDeviceWindow              = hWnd;
d3dpp.Windowed                   = true;
d3dpp.EnableAutoDepthStencil     = true;
d3dpp.AutoDepthStencilFormat     = D3DFMT_D24S8;
d3dpp.Flags                      = 0;
d3dpp.FullScreen_RefreshRateInHz = D3DPRESENT_RATE_DEFAULT;
d3dpp.PresentationInterval = D3DPRESENT_INTERVAL_IMMEDIATE;

if (FAILED(g_pD3D->CreateDevice(D3DADAPTER_DEFAULT,
                                 D3DDEVTYPE_HAL, hWnd, vp,
                                  &d3dpp, &g_pd3dDevice))){
    MessageBox(NULL, "CreateDevice()", "Error", NULL);
        g_bDone = true;
}

ShowWindow(hWnd, SW_SHOW);
```

STIKOM SURABAYA

Sedangkan dengan menggunakan framework rendering engine, inisialisasi hanya cukup dengan membuat obyek rendering dan memanggil fungsi inisialisasinya saja. Untuk pengaturan device dilakukan pada saat form inisialisasi tampil. Berikut adalah contoh kodenya:

```
// INISIALISASI

g_pRenderer = new TumozRenderer(hInst);

if (FAILED(g_pRenderer->CreateDevice(API_DIRECT3D))) {
        MessageBox(NULL, "g_pRenderer->CreateDevice()",
                 "Error", NULL);
        g_bDone = true;
}

g_pDevice = g_pRenderer->GetDevice();
if (g_pDevice == NULL) {
        MessageBox(NULL, "Device Tidak Ada", "Error", NULL);
        g_bDone = true;
}

if (FAILED(g_pDevice->Init(hWnd, NULL, 0, 16, 8, true))) {
        return 0;
}

ShowWindow(hWnd, SW_SHOW);
```

Berdasarkan hasil uji coba terhadap proses inisialisasi diatas maka dapat disimpulkan bahwa menggunakan API DirectX memiliki proses yang lebih panjang dan pengembang diharapkan untuk mengerti konsep sistem pipeline kartu grafis yang kompleks seperti pemahaman konsep buffer, stencil, kemampuan hardware pemrosesan verteks grafis dan lain-lain. Dan dengan adanya framework rendering engine yang lebih sederhana pengembang dapat lebih fokus untuk memulai pembuatan aplikasi 3D dan menyerahkan otomatisasi pengaturan sistem pada framework rendering engine.

4.4.2. Pengujian Perbandingan Pembuatan Obyek 3D

Pembuatan obyek font menggunakan API DirectX harus menggunakan kelas enumerasi untuk pengaturannya seperti ukuran, ketebalan, jenis, dan sebagainya. Lalu fungsi pembuatan font di panggil berdasarkan pengaturan yang

STIKOM SURABAYA

telah dibuat sebelumnya. Sedangkan pada framework rendering engine untuk membuat obyek font hanya dengan memanggil fungsi pembuatan font beserta parameternya.

- Pembuatan obyek font menggunakan API DirectX:

```
// BUAT OBYEK FONT

D3DXFONT_DESC fDesc;

HDC hDC = GetDC( NULL );
int nHeight = -MulDiv(20, GetDeviceCaps(hDC, LOGPIXELSY),
                        72);

fDesc.Height = nHeight;
fDesc.Width  = 0;
fDesc.Weight = 0;
fDesc.Italic = false;
fDesc.CharSet = DEFAULT_CHARSET;
fDesc.OutputPrecision = OUT_DEFAULT_PRECIS;
fDesc.Quality = DEFAULT_QUALITY;
fDesc.PitchAndFamily = DEFAULT_PITCH | FF_DONTCARE;
strcpy_s(fDesc.FaceName, "Arial");

ID3DXFont* pFont = NULL;
if(FAILED(D3DXCreateFontIndirect(g_pd3dDevice, &fDesc,
                                     &pFont))) {
       MessageBox(NULL, "D3DXCreateFontIndirect()", "Error",
                   NULL);
       g_bDone = true;
}
```

- Pembuatan obyek font menggunakan framework rendering engine:

```
// BUAT OBYEK FONT

if (FAILED(g_pDevice->CreateFont("Arial", 0, false, false,
                                     false, 40, &g_nFontID))) {
       MessageBox(NULL, "Buat Font gagal", "Error", NULL);
       g_bDone = true;
}
```

Proses pembuatan obyek segitiga pada API DirectX memerlukan beberapa proses yaitu diawali dengan membuat deklarasi format fleksibel verteksnya terlebih dahulu seperti di bawah ini:

```
struct CUSTOMVERTEX {
       FLOAT    x, y ,z;
       D3DCOLOR color;
       FLOAT    tu, tv;
};

#defineD3DFVF_CUSTOMVERTEX (D3DFVF_XYZ|D3DFVF_DIFFUSE|D3DFVF_TEX1)
```

STIKOM SURABAYA

Setelah itu membuat verteks buffer dan memasukkan data verteks tersebut dengan menggunakan metode penguncian (*locking*) seperti kode di bawah ini:

```
// BUAT OBYEK SEGITIGA

if(FAILED(g_pd3dDevice->CreateVertexBuffer(3 *
                                    sizeof(CUSTOMVERTEX),
                                    0, D3DFVF_CUSTOMVERTEX,
                                    D3DPOOL_DEFAULT, &g_pVB,
                                    NULL))){
        MessageBox(NULL, "CreateVertexBuffer()", "Error",
                   NULL);
        g_bDone = true;
}

CUSTOMVERTEX* v;

if( FAILED(g_pVB->Lock(0, 0, (void**)&v, 0))){
        MessageBox(NULL, "Lock()", "Error", NULL);
        g_bDone = true;
 }

        v[0].x       =  0.0f;
        v[0].y       =  5.0f;
        v[0].z       =  0.0f;
        v[0].color   = D3DCOLOR_RGBA(255, 255, 255, 255);
        v[0].tu      =  0.0f;
        v[0].tv      =  1.0f;


        v[1].x       =  5.0f;
        v[1].y       = -5.0f;
        v[1].z       =  0.0f;
        v[1].color   = D3DCOLOR_RGBA(255, 255, 255, 255);
        v[1].tu      =  1.0f;
        v[1].tv      = -1.0f;

        v[2].x       = -5.0f;
        v[2].y       = -5.0f;
        v[2].z       =  0.0f;
        v[2].color   = D3DCOLOR_RGBA(255, 255, 255, 255);
        v[2].tu      = -1.0f;
        v[2].tv      = -1.0f;

    g_pVB->Unlock();
```

Berbeda dengan sistem dari framework rendering engine, untuk membuat obyek segitiga diawali dengan membuat array dari kelas enumerasi verteks yang telah disediakan (ada 6 pilihan jenis verteks) sehingga user tidak perlu membuat deklarasi verteksnya sendiri. Kemudian verteks tersebut diisi datanya dan dipanggil fungsi pembuatannya sebagai berikut:

```
// BUAT OBYEK SEGITIGA

VERTEX v[3];
```

STIKOM SURABAYA

```
WORD   i[3] = {0,  1,  2};

memset(v, 0, sizeof(VERTEX)*3);

v[0].x       =  0.0f;
v[0].y       =  5.0f;
v[0].z       =  0.0f;
v[0].tu      =  0.0f;
v[0].tv      =  1.0f;

v[1].x       =  5.0f;
v[1].y       = -5.0f;
v[1].z       =  0.0f;
v[1].tu      =  1.0f;
v[1].tv      = -1.0f;

v[2].x      = -5.0f;
v[2].y      = -5.0f;
v[2].z      =  0.0f;
v[2].tu      = -1.0f;
v[2].tv      = -1.0f;

if (FAILED(
   g_pDevice->GetVertexManager()->CreateStaticBuffer(VID_UU,
                              0, 3, 3, v, i, &g_nTriangle))) {
       MessageBox(NULL, "Buat segitiga gagal", "Error",NULL);
       g_bDone = true;
}
```

Apabila obyek segitiga telah dibuat, maka perlu juga dibuatkan teksturnya dalam hal ini API DirectX memanggil fungsi pembuatan teksturnya berserta pengaturan *sampler* dan texture stagenya. Sedangkan framework rendering engine menggunakan skin manajer untuk pengaturan tekstur dan warna material dari obyek segitiga tersebut.

- Pembuatan tekstur menggunakan API DirectX:

```
// BUAT TEKSTUR

if( FAILED(D3DXCreateTextureFromFile(g_pd3dDevice,
                                         "texture.bmp",
                                         &g_pTexture))) {
       MessageBox(NULL, "D3DXCreateTextureFromFile()",
                  "Error", NULL);
       g_bDone = true;
}

g_pd3dDevice->SetTexture(0, g_pTexture);

g_pd3dDevice->SetSamplerState(0, D3DSAMP_MAGFILTER,
                              D3DTEXF_LINEAR);
g_pd3dDevice->SetSamplerState(0, D3DSAMP_MINFILTER,
                              D3DTEXF_LINEAR);
g_pd3dDevice->SetSamplerState(0, D3DSAMP_MIPFILTER,
                              D3DTEXF_POINT);
```

STIKOM SURABAYA

```
g_pd3dDevice->SetTextureStageState(0, D3DTSS_COLOROP,
                                   D3DTOP_MODULATE);
g_pd3dDevice->SetTextureStageState(0, D3DTSS_COLORARG1,
                                   D3DTA_TEXTURE);
g_pd3dDevice->SetTextureStageState(0, D3DTSS_COLORARG2,
                                   D3DTA_DIFFUSE);
g_pd3dDevice->SetTextureStageState(0, D3DTSS_ALPHAOP,
                                   D3DTOP_DISABLE);
```

- Pembuatan tekstur menggunakan framework rendering engine:

```
// BUAT TEKSTUR

UINT      s=0;
TUMOZCOLOR c = { 1.0f, 1.0f, 1.0f, 1.0f };

g_pDevice->GetSkinManager()->AddSkin(&c, &c, &c, &c, 1.0f,
                                     &s);
g_pDevice->GetSkinManager()->AddTexture(s, "texture.bmp",
                                        false, 0, NULL, 0);
```

Dapat dilihat dalam hal pembuatan tekstur diatas bahwa framework rendering engine menggunakan skin manajer untuk mengelola pengaturan kondisi yang berhubungan dengan semua tekstur tersebut seperti sampler, texture stage , dan pengaturan lainnya.

.

4.4.3. Pengujian Perbandingan Rendering Obyek 3D

Untuk merender obyek 3D (dalam hal ini obyek segitiga) pada API DirectX diperlukan 3 fungsi utama yaitu pertama diawali dengan memanggil fungsi untuk mengaitkan verteks buffer ke stream data, yang utamanya untuk mengisi data geometri ke dalam rendering pipeline. Kedua memanggil fungsi untuk mengatur format verteks yang dideklarasikan sebelumnya untuk memberitahu pipeline jenis tipe format dari verteks. Dan yang ketiga adalah memanggil fungsi untuk mengirim data primitif tersebut (titik, garis, atau polygon segitiga) ke dalam pipeline agar dapat di render. Berikut ini adalah contoh kode sumbernya:

```
// RENDER OBYEK SEGITIGA

g_pd3dDevice->SetStreamSource(0, g_pVB, 0,
```

STIKOM SURABAYA

```
                                  sizeof(CUSTOMVERTEX));
g_pd3dDevice->SetFVF(D3DFVF_CUSTOMVERTEX);
g_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLELIST, 0, 1);
```

Framework rendering engine memiliki sistem khusus untuk menganalisa data yang akan di render agar dapat sesuai dengan sistem rendering pipeline pada API grafis yaitu menggunakan verteks manajer, sehingga untuk merender obyek segitiga pada contoh dibawah ini hanya memanggil dengan satu fungsi render.

```
// RENDER OBYEK SEGITIGA

if (FAILED(g_pDevice->GetVertexManager()->Render(
                                          g_nTriangle))) {
      MessageBox(NULL, "Render segitiga gagal",
                 "Error", NULL);
}
```

Perbedaan pembuatan obyek font menggunakan API DirectX dengan framework rendering engine adalah bahwa API DirectX harus menentukan terlebih dahulu posisi penempatannya menggunakan kelas segiempat layar seperti kode di bawah ini:

```
// RENDER OBYEK FONT

RECT rc = { 10, 10, 0, 0 };

pFont->DrawText(NULL, "Testing.....!", -1, &rc,
                DT_SINGLELINE,D3DCOLOR_ARGB(255, 255, 255,
                0));
```

Sedangkan framework rendering engine menggunakan fungsi render teks yang lebih sederhana dari render device seperti berikut ini:

```
// RENDER OBYEK FONT

g_pDevice->DrawText(g_nFontID, 10, 10, 255, 255, 0,
                    "Testing.....!");
```

4.5. Pengujian Implementasi Manajemen Scene

Sistem manajemen scene merupakan gabungan dari penerapan algoritma, perhitungan 3D (ray, plane, AABB, dsb) dan sistem kendali berdasarkan waktu, kamera dan kontrol pregerakan pada saat proses rendering berjalan. Tujuan dari

STIKOM SURABAYA

manajemen scene adalah untuk meningkatkan performa scene seefisien mungkin agar dapat berjalan dengan lancar pada saat rendering berdasarkan keterbatasan kemampuan hardware grafis yang ada.

Untuk mengimplementasikan manajemen scene dimulai dengan memuat semua polygon yang akan di proses dan memasukkannya ke dalam algoritma manajemen scene (baik Octree maupun BSP Tree) dan kemudian membangun strukturnya (membuat simpul). Berikut ini adalah potongan kodenya:

```
// buka file level
FILE *pFile = fopen(chFile, "rb");
if (!pFile) return false;

// baca jumlah dari polygon yang datang
fread(&Num, sizeof(UINT), 1, pFile);
if (Num == 0) return false;

// alokasi memory
pList = new TumozPolygon[Num];

// memuat semua polygon
for (UINT i=0; i<Num; i++) {

       // baca nama dari texture basis
       fread(&n, sizeof(UINT), 1, pFile);
       fread(buffer, n, 1, pFile);
       buffer[n] = '\0';

       // memuat polygon dan salin ke list
       LoadPolygon(pFile, &Poly);
       pList[i].CopyOf(Poly);
} // for

// akhirnya buat BSP tree
g_pBspTree = new TumozBSPTree;
g_pBspTree->BuildTree(pList, Num);

// akhirnya buat octree
g_pOctree = new TumozOctree;
g_pOctree->BuildTree(pList, Num);
```

Setelah algoritma manajemen scene telah di buat, maka dilanjutkan dengan memproses obyek timer untuk mengupdate siklus rendering dan obyek kontrol pergerakan kamera (dalam hal ini kamera sudut pandang pertama) untuk menentukan orientasi dan sudut pandang area scene yang akan ditampilkan ke layar seperti yang dicontohkan pada potongan kode di bawah ini:

STIKOM SURABAYA

```
// update dan reset controller
g_pTimer->Update();

// ambil posisi saat ini dan gunakan movement
TumozVector vcOld = g_pMCEgo->GetPos(), vcNew(0, 0, 0);
g_pMCEgo->Update(g_pTimer->GetElapsed());
g_pMCEgo->SetSpeed(0.0f);
g_pMCEgo->SetSlideSpeed(0.0f);
g_pMCEgo->SetRotationSpeedX(0.0f);
g_pMCEgo->SetRotationSpeedY(0.0f);

// update posisi movement controller
g_pDevice->SetView3D(g_pMCEgo->GetRight(),
                        g_pMCEgo->GetUp(),
                        g_pMCEgo->GetDir(),
                        g_pMCEgo->GetPos());
```

Setelah itu algoritma manajemen scene akan melakukan traversal (proses menjelajahi simpul untuk mengambil polygon yang dibutuhkan) yang ada di dalam *frustum* (ruang lingkup kamera) dan setelah itu daftar polygon tersebut di render dan ditampilkan di layar. Berikut ini adalah potongan kodenya:

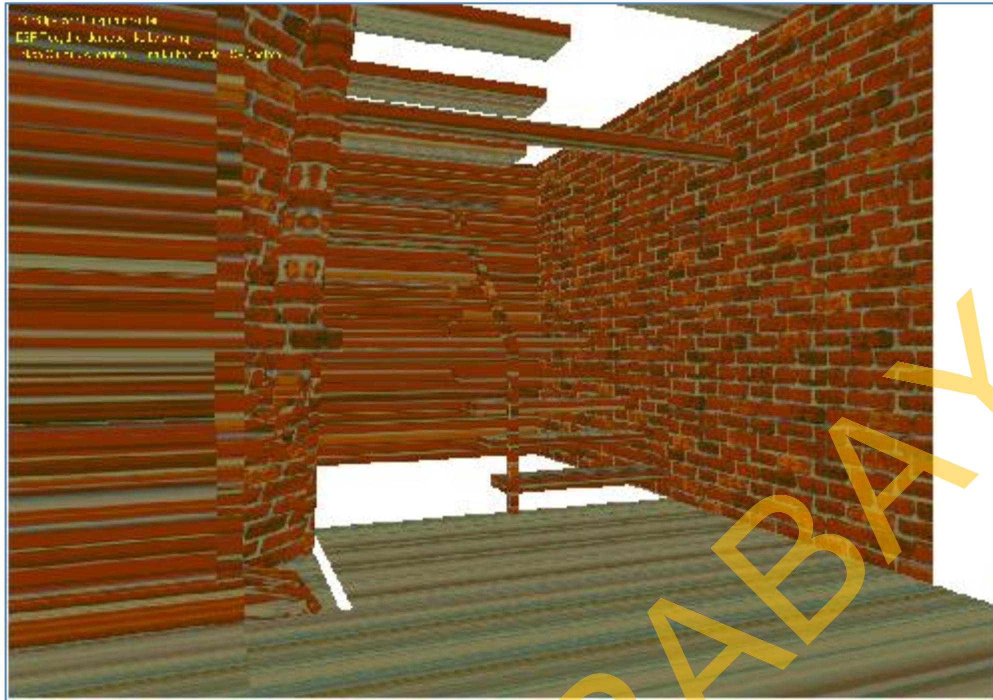
```
// ambil view frustum saat ini
g_pDevice->GetFrustrum(Frustum);

// traverse tree
if ( g_bBsp ) {
       if (g_bFtB)
              // traverse depan ke belakang
              g_pBspTree->TraverseFtB(&List,
                                   g_pMCEgo->GetPos(), Frustum);
       else
              // traverse belakang ke depan
              g_pBspTree->TraverseBtF(&List,
                                   g_pMCEgo->GetPos(), Frustum);
}
else {
       g_pOctree->Traverse(&List, &AabbList, Frustum);
}

RenderPolylist(List, g_bWired, &clrG);
```

Seperti yang terlihat pada gambar 4.9, sistem manajemen scene hanya memproses rendering polygon dari dalam area sudut pandang kamera saja dan sedangkan polygon di luar area tersebut tidak akan diproses. Dengan demikian akan meningkatkan kinerja dari hardware grafis sehingga menghasilkan nilai frame rate per second (fps) yang cukup signifikan dari proses rendering.

STIKOM SURABAYA



Gambar 4.9. Manajemen Scene Ruang Interior

Berdasarkan hasil analisa uji coba sistem manajemen scene tersebut terdapat hasil yang berbeda antara menggunakan algoritma BSP Tree dengan Octree. Dengan menggunakan program demo, algoritma Octree menghasilkan polygon lebih banyak dan fps lebih kecil daripada BSP Tree. Sehingga dapat disimpulkan bahwa algoritma BSP Tree jauh lebih efisien daripada Octree. Selain itu hasil pengujian juga menunjukkan peningkatan hasil fps yang cukup signifikan dari penggunaan teknologi SIMD yaitu sebesar antara 24% sampai dengan 40%. Berikut ini adalah tabel hasil pengujian manajemen scene:

Tabel 4.1. Tabel Pengujian Algoritma Manajemen Scene

Ket	Octree				BSP Tree			
	Jml Polygon	Rata-Rata FPS		Kenaikan FPS (%)	Jml Polygon	Rata-Rata FPS		Kenaikan FPS (%)
		Non SIMD	SIMD			Non SIMD	SIMD	
Maks. View	1458	7,44	9,23	24,06	498	47,97	60,22	25,54
Min. View	63	223,93	315,21	40,76	20	267,28	343,61	28,56

STIKOM SURABAYA

4.6. Evaluasi Sistem

Agar dapat mengetahui bahwa fungsi-fungsi dari framework rendering engine berjalan dengan baik, tanpa adanya kendala atau error, dan berjalan sesuai dengan harapan, maka perlu dilakukan analisa pengujian sistem. Kelebihan dan kelemahan dari framework ini akan di uji dan di evaluasi sebelum dapat diimplementasikan secara nyata. Masing-masing proses akan di uji, dan pengujian yang dilakukan adalah sebagai berikut:

4.6.1. Hasil Uji Coba Inisialisasi Framework Rendering Engine

Uji coba insialisasi framework rendering engine dilakukan dengan inputan window yang telah di buat oleh pihak pengembang untuk pembuatan perangkat rendering. Hasil uji coba dapat dilihat pada tabel 4.2.

Tabel 4.2. Hasil Uji Coba Inisialisasi Framework Rendering Engine

Test Case	Tujuan	Input	Output yang diharapkan	Status
1	Melakukan inisialisasi dengan menampilkan form pemilihan adapter dan format	Handle windows utama, array child windows, dan jumlah array child windows	Menampilkan form untuk memilih adapter, format tampilan layar, dan sebagainya, lalu setelah itu menampilkan proses rendering	Sukses
2	Melakukan inisialisasi tanpa menampilkan form pemilihan adapter dan format	Handle windows utama, array child windows, jumlah array child windows, dan setting adapter & format	Langsung menampilkan proses rendering	Sukses

STIKOM SURABAYA

4.6.2. Hasil Uji Coba Pengaturan Scene

Uji coba pengaturan scene melakukan bermacam-macam fungsi dari perangkat rendering yang telah aktif. Hasil pengujian dapat dilihat pada tabel 4.3.

Tabel 4.3. Hasil Uji Coba Pengaturan Scene

Test Case	Tujuan	Input	Output yang diharapkan	Status
3	Memilih target windows yang akan memproses rendering	Indeks array child windows	Menampilkan windows yang dipilih untuk proses rendering	Sukses
4	Memilih viewport stage untuk memproyeksikan scene ke layar	Nilai FOV, dimensi layar, dan indeks stage	Menampilkan scene berdasarkan efek zoom FOV	Sukses
5	Mengatur arah pandang atau kamera pada scene	Vektor kanan, vektor atas, vektor arah, dan vektor posisi kamera	Menampilkan pandangan atau view kamera terhadap scene	Sukses
6	Membersihkan layar atau memberi warna background pada scene	nilai warna merah, hijau, biru, dan alpha untuk transparansi	Warna background scene akan berubah sesuai warna yang ditentukan	Sukses

4.6.3. Hasil Uji Coba Rendering Obyek Tiga Dimensi

Pada uji coba ini, pihak pengembang membuat obyek tiga dimensi dan melakukan proses rendering pada obyek tiga dimensi tersebut. Hasil uji coba dapat dilihat pada tabel 4.4.

Tabel 4.4. Hasil Uji Coba Rendering Obyek Tiga Dimensi

Test Case	Tujuan	Input	Output yang diharapkan	Status
7	Membuat Skin dan tekstur obyek tiga dimensi	4 structure TUMOZCOLOR, dan file tekstur (.BMP)	ID skin yang telah di buat	Sukses

STIKOM SURABAYA

8	Membuat obyek tiga dimensi berdasarkan verteksnya	Enumeration TUMOZVERTEXID, array verteks, dan array indeks	ID obyek tiga dimensi yang telah di buat	Sukses
9	Melakukan rendering obyek tiga dimensi pada scene	ID obyek tiga dimensi	Menampilkan gambar obyek tiga dimensi pada scene	Sukses

4.6.4. Hasil Uji Coba Transformasi dan Pencahayaan

Hasil uji coba proses transformasi dan pencahayaan untuk pergerakan dinamis dan beragam efek pencahayaan dapat dilihat pada tabel 4.5.

Tabel 4.5. Hasil Uji Coba Transformasi dan Pencahayaan

Test Case	Tujuan	Input	Output yang diharapkan	Status
10	Melakukan proses transformasi obyek tiga dimensi pada scene	TumozMatrix yang merupakan hasil gabungan transformasi pada seluruh scene	Menampilkan transformasi dan pergerakan obyek tiga dimensi pada saat rendering	Sukses
11	Melakukan efek pencahayaan secara merata pada scene	3 nilai float untuk merah, hijau, dan biru	Scene akan diterangi oleh cahaya secara merata	Sukses
12	Melakukan beragam efek pencahayaan secara khusus pada stage tertentu	Structure TUMOZLIGHT, dan indeks stage yang di tuju	Scene akan diterangi bermacam-macam efek pencahayaan seperti efek cahaya lampu sorot, dsb	Sukses

STIKOM SURABAYA

4.6.5. Hasil Uji Coba Implementasi Shader

Hasil uji coba implementasi shader untuk pemrograman vertex dan pixel shader dapat dilihat pada tabel 4.6.

Tabel 4.6. Hasil Uji Coba Implementasi Shader

Test Case	Tujuan	Input	Output yang diharapkan	Status
13	Membuat obyek vertex dan pixel shader	Nama file atau stream data shader, boolean file, dan boolean kompilasi	ID shader yang telah di buat	Sukses
14	Mengatur variabel konstan untuk membantu proses shader	Enumeration TUMOZSHADERTYPE, TUMOZDATATYPE, indeks register awal, jumlah array data, dan data array nilai variabel	Menyediakan variabel konstan yang akan digunakan pada proses shader	Sukses
15	Mengaktifkan shader pada saat proses rendering	ID shader yang telah dibuat, dan enum TUMOZVERTEXID untuk verteks shader	Beragam efek shading pada proses rendering	Sukses

STIKOM SURABAYA