Tutorial VRML Bagian 1:

Membuat Objek 3D dengan VRML

VRML adalah salah satu bahasa komputer untuk membuat model objek 3 dimensi dalam dunia virtual. Tidak hanya itu, untuk mensimulasi objek yang bergerakpun dapat dilakukan dengan VRML. Walau sebelumnya diperkirakan akan mati, tapi ternyata makin banyak aplikasi lain yang mengintegrasikan dukungan format file VRML. Harus diakui masih sedikit tool authoring yang mendukung penggenerasian VRML secara GUI. Mungkin ini salah satu sebab yang membuat orang jarang memakai VRML bagi proyek multimedianya. Namun di balik itu, di bidang pendidikan ternyata banyak proyek-proyek yang memanfaatkan VRML guna melakukan animasi atau verifikasi secara visual. Ini tentu saja tidak lepas dari multifungsi VRML, yang salah satunya memudahkan presentasi lewat media internet. Selain itu kemampuan VRML versi 2.0-nya yang mendukung bahasa skrip seperti ECMAScript (skrip VRML), Javascript dan Java memegang peranan sangat penting.

Isi dari tutorial ini lebih ditekankan pada fitur yang ada pada VRML serta pengertian tentang cara kerjanya. Bagaimana cara membangun dunia virtual dilihat dari nilai seni dan tata layoutnya tidak tercakup di sini. Ini memang disebabkan background penulis adalah orang teknik.

Seri tutorial ini akan dibagi menjadi 4 bagian. Artikel bagian pertama, yang sedang Anda baca ini, akan membahas resource dan contoh proyek yang saat ini sedang aktual. Lalu dasar-dasar dari VRML, seperti konsep dan struktur VRML, cara membuat objek geometri dasar, serta cara memanipulasi tampakan dari objek ini. Tidak kalah pentingnya adalah mengerti sistem koordinat yang dipakai oleh VRML.

Tema utama dalam bagian kedua nanti akan membahas tentang konsep even dan cara menyambungnya (event routing). Dengan disertai contoh yang sederhana, beberapa sumber yang bisa menyebabkan even akan ditampilkan. Tentu saja cara untuk mengirimkan dan menangkap even juga akan dibahas di sini.

Pada bagian ketiga, akan dibahas tema yang lebih kompleks, yaitu cara membuat animasi dengan memanfaatkan bahasa skrip. Karena even adalah satu-satunya cara untuk berkomunikasi, maka di sinipun digunakan metode even seperti yang dibahas pada bagian sebelumnya.

Bagian terakhir dari tutorial ini akan membahas cara mengintegrasikan VRML dengan bahasa Java. Biasanya ini dilakukan pada proyek yang membutuhkan algoritma dan perhitungan yang kompleks. Contohnya adalah untuk membuat simulasi lengan robot, robot berjalan dan sebagainya.

VRML? Apa itu?

Kalau Anda belum ngeh apa itu VRML, mungkin singkatannya dapat memberi sedikit ide, Virtual Reality Modeling Language, di mana untuk pertama kalinya tahun 1994 VRML versi pertamanya diperkenalkan. VRML, seperti halnya HTML adalah bahasa skrip dalam format teks polos (`ASCII` pada versi 1.0 ataupun `utf8` pada versi 2.0). Bedanya VRML digunakan

untuk menggambarkan scene 3 dimensi dalam ruangan virtual. Disebut ruangan virtual karena kita seakan-akan berada dalam ruangan yang bisa melihat objek 3D dari sudut pandang yang kita inginkan, tidak dengan berjalan kaki, tapi dengan memanfaatkan interface komputer dalam berinteraksi dengan monitor, seperti contohnya dengan menggunakan mouse. Namun dalam versi pertamanya ini semua objek geometri masih statik, tidak mempunyai kemungkinan bergerak. Interaksi dinamis antara user dengan objek masih terbatas, komunikasi antara objek satu dengan lainnya juga belum dimungkinkan. Kelemahan ini segera teratasi dengan dikeluarkannya spesifikasi baru VRML 2.0 pada tahun 1996. Dengan versi baru ini, maka terbuka kemungkinan baru bagi penggunaan VRML, seperti yang banyak digunakan adalah animasi. Versi ini sekarang juga disebut dengan standar ISO VRML97 yang merupakan format file grafik 3D standar untuk aplikasi Internet maupun WWW. Secara prinsip tidak ada perbedaan antara VRML 2.0 dengan VRML97, sehingga kita akan menyebutnya VRML97, dan spesifikasi inilah yang akan digunakan sebagai referensi dalam artikel ini.

Sebenarnya banyak proyek yang memanfaatkan VRML, hal ini dapat dilihat di situs Web konsorsium Web3D, www.vrml.org. Kebanyakan VRML ini tidak digunakan sebagai satu-satunya bahasa pemrograman, melainkan dipadukan dengan tool lainnya. Salah satu contohnya adalah proyek Interactive Robot Manipulation with VRML 2.0 yang dilakukan oleh DLR (German Eurospace Center) yang memadukan VRML dengan Java3d Robots. Contoh satu lagi adalah proyek Autonome Walking di Universitas Duisburg yang menggunakan VRML sebagai visualisasi dari tools Walking Pattern Generator yang ditulis dengan bahasa C++.

Untuk sekedar diketahui, VRML adalah salah satu teknik pemrograman objek 3D interaktif di Internet disamping Java3D dan X3D di bawah bendera konsorsium Web3D.

Browser VRML

Sebelum kita mulai belajar menggunakan VRML-nya sendiri, ada baiknya kita tahu browser apa saja yang dapat digunakan untuk menampilkan VRML. File VRML biasanya mempunyai akhiran `wrl`, berasal dari **world**. Dalam bentuk terkompresi, akhiran ini bisa juga lain seperti `wrz` atau `wrl.gz`. Ini tidak bermasalah, sebab biasanya browser juga mendukung jenis file yang terkompresi. Untuk dapat menampilkan isi file ini, tentu saja dibutuhkan aplikasi yang bisa menginterpretasikan bahasa VRML. Dalam praktiknya, aplikasi ini sebagian besar dibuat dalam bentuk plugin yang berjalan di bawah kontrol browser seperti IE, Netscape, ataupun Opera. Situs Web www.vrml.org menyediakan banyak resource untuk VRML, termasuk browser dan plugin untuk VRML. Dalam bentuk plugin dapat disebutkan antara lain **Blaxxun Contact**, **BlenderWeb Plugin**, **Cosmo Player**, **Viscape**, dll. Biasanya plugin tersebut tidak hanya mampu untuk menampakkan VRML, melainkan juga format 3D lainnya.

Selain dalam bentuk plugin tentu saja ada juga browser dalam bentuk aplikasi standalone. Contohnya antara lain, **OpenVRML**, **FreeWRL**, **VRWeb**, **VRMLView** dari SIM, **Cortona VRML,** dan sebagainya. Beberapa aplikasi komersial seperti **Open Inventor** dari TSG bahkan telah mendukung VRML, tidak hanya dalam level konversi antarformat file, tapi sebagai tool authoring untuk VRML.

Gambar-gambar yang ditampilkan dalam artikel ini sendiri dicapture dari plugin Blaxxun Contact yang dapat didownload dari ftp.blaxxun.com. Sayang software ini hanya berjalan di

sistem operasi Windows. Bagi pemakai Unix dapat mencoba OpenVRML atau FreeWRL. OpenVRML sebenarnya adalah tools untuk pembuatan aplikasi VRML dengan menyertakan contoh program browser VRML yang masih sederhana.

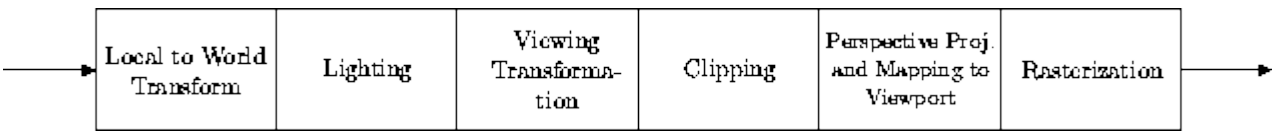
Lalu Bagaimana Menghasilkan File VRML?

Seperti disinggung sebelumnya, isi dari file VRML adalah berupa teks yang dapat dibaca dan dimengerti oleh kita secara langsung (human readable). Maka pada prinsipnya semua editor teks dapat digunakan untuk membangun isi file ini. Bagi yang ingin lebih profesional, dapat menggunakan editor yang dibuat khusus untuk mengedit file VRML, yaitu **VrmlPad** yang bisa didapatkan dari situs www.parallelgraphics.com/products/vrmlpad. Selain *kita* (dengan cara "doing it by hand"), dapat juga digunakan program aplikasi untuk menggenerasi file VRML, biasanya disebut tool authoring VRML. Salah satunya adalah Open Inventor dari TGS, seperti telah disinggung sebelumnya, yang menyediakan tool grafik C++ yang konon paling banyak digunakan di dunia untuk menggenerasi node dari VRML. Ini tidak mengherankan, karena dilihat dari sejarahnya, format yang dipakai oleh VRML adalah hasil tiruan dari format Open Inventor, dengan berprinsip pada *don't reinvent the wheel*. Selain itu **Pro/E**, salah satu program aplikasi yang digunakan untuk mengkonstruksi di bidang teknik mesin dapat juga menghasilkan file berformat VRML. Namun pada **ProE/2000i**, hasil eksportnya masih berformat VRML 1.0.

Bahasan dalam artikel ini hanya terbatas cara menggunakan VRML secara manual untuk membangun dunia virtual. Oleh karena itu, sangat penting sekali mengerti cara kerja tiap perintah dari VRML. Salah satu keuntungan cara manual ini adalah, kita dapat membuat dunia virtual dengan VRML secara optimal, baik dari ukuran besar file maupun strukturnya. Tentu saja ini disertai dengan nilai minusnya, seperti tingkat kerumitan yang makin meningkat dengan makin kompleksnya objek yang kita inginkan.

Dasar Dari Pemrosesan Grafik

Walau kita tidak akan belajar menditel tentang pemrosesan grafik 3D, tidak ada salahnya kita tahu sedikit apa yang sebenarnya terjadi saat objek 3D ini digenerasi oleh komputer kita. Setiap kali isi VRML harus digambarkan ke tampilan, maka ada proses tertentu yang harus dikerjakan. Urutan dari proses ini disebut dengan Graphics Pipeline yang diagramnya dapat dilihat di Gambar 1 (diambil dari www.uark.edu/ wrgx/vrml/). Bergantung pada tool yang digunakan, maka proses ini dapat diimplementasikan dengan software atau mungkin sudah diimplementasikan langsung di kartu grafiknya. Tool yang digunakan untuk mengimplementasi Graphics Pipeline ini biasanya disebut juga dengan rendering engine. Beberapa jenis rendering engine ini al: RealityLab, RenderWare, 3DR, OpenGL, Mesa, Quickdraw 3D.



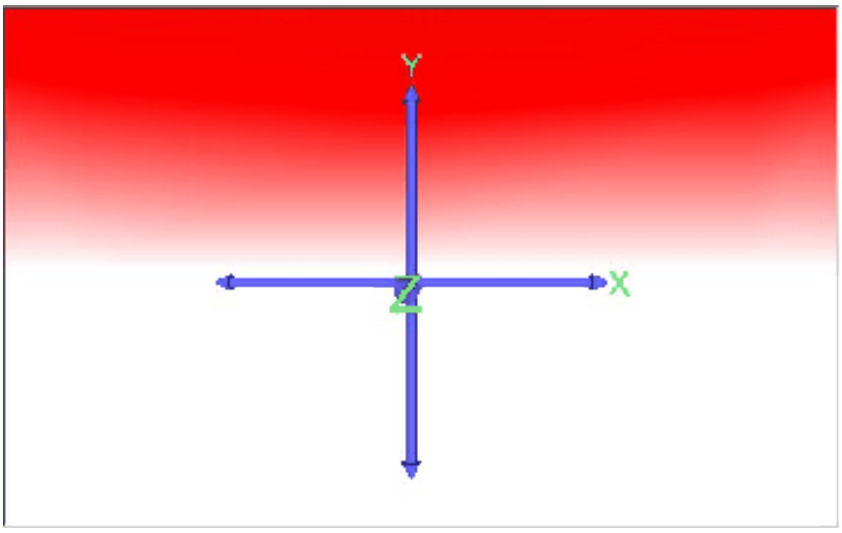
Gb 1. Graphics Pipeline

Apa maksud setiap bagian proses di atas, akan secara singkat diterangkan seperti berikut:

- Local to World Transforms. Biasanya objek dibangun dengan memakai sistem koordinat lokal sebagai referensinya. Karena untuk melakukan perhitungan harus dipakai sistem referensi yang sama, maka sistem koordinat lokal ini harus ditransformasikan ke sistem koordinat referensi, biasanya disebut dengan sistem koordinat dunia.
- Lighting. Di sini dilakukan perhitungan tampakan dari permukaan objek sebagai hasil dari pencahayaan, seperti warna, transparensi, refleksi dan sebagainya.
- Viewing Transformation. Di sini titik pojok dari objek akan ditransformasi dari sistem koordinat dunia ke koordinat kamera.
- Clipping. Seperti namanya, di sini bagian yang tidak nampak akan dipotong dan tidak diproses.
- Perspective Projection and Mapping to Viewport. Untuk medapatkan kesan 3D, maka di sini dilakukan proyeksi dengan memasukkan perspektif. Hasilnya akan ditampilkan ke koordinat display.
- Rasterization. Setelah perhitungan di atas selesai, maka semua objek harus ditampilkan ke layar monitor. Di sini nilai warna setiap titik pada raster akan ditentukan.

Sistem Koordinat

Salah satu yang sangat penting dalam membuat dunia VRML adalah mengerti cara kerja sistem koordinat yang digunakan. Sistem koordinat ini dijadikan referensi setiap kali besaran gerak dilakukan, baik oleh objek geometri maupun ketika kita bernavigasi. Di sini kata "besaran gerak" jangan hanya diartikan sebagai besaran yang berpindah, melainkan yang diam pun juga punya besaran gerak. Termasuk dalam besaran gerak di sini antara lain posisi, orientasi, gerak translasi maupun rotasi. VRML menggunakan sistem koordinat kartesius, right-handed, *x*,*y*,*z*. Sumbu *x* mempunyai arah positif ke kanan, sedang arah positif dari sumbu *y* adalah ke atas. Sumbu *x* dan *y* ini membelah window browser menjadi dua bagian yang simetri, kiri dengan kanan dan atas dengan bawah. Titik origin dari kedua sumbu tersebut berada tepat di pusat browser. Sedang sumbu *z* mempunyai arah positif dari monitor ke arah mata kita. Gambar 2 mengilustrasikan sistem koordinat ini.



Gb 2. Sistem koordinat yang dipakai di VRML

Bagaimana Dengan Satuan?

Kalau kita hendak membuat suatu objek geometri, tentu saja kita harus memberikan satuannya. Pada ukuran panjang, satuannya tidaklah mengambil peranan penting selama kita selalu konsisten dengan skalanya. Namun pada kasus tertentu ada kalanya kita harus benar-benar memperhatikan satuannya ini. VRML mempunyai satuan standar yang dipakai dalam membangun dunia virtual, seperti terlihat pada Tabel 1.

Bagi yang belum terbiasa bekerja dengan satuan radian, dapat digunakan persamaan berikut untuk menghitung nilai sudut dari derajat ke radian.

```
x [radian] = y [derajat] * 3.14 / 180
```

di mana y adalah nilai dalam derajat, dan x adalah nilai yang dicari dalam radian.

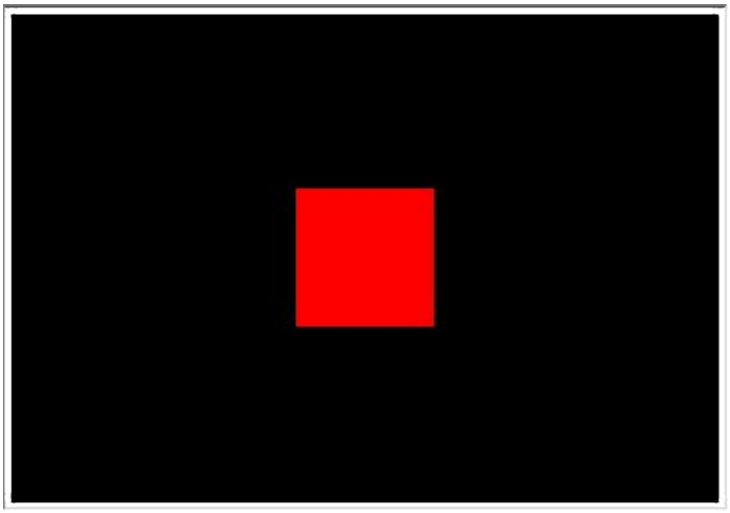
Hirarki dan Struktur

Dunia VRML terbangun dari struktur yang terhirarki. File VRML sendiri adalah hirarki yang paling atas, yang terdiri dari header, scene-graph, prototipe dan event routing. Struktur yang paling bawah adalah node, yang terdiri dari field yang berisi properti dan informasi tentang node yang memilikinya. Secara garis besar, kumpulan dari node ini akan membangun scene-graph. Namun harus diingat, keterangan di atas hanyalah untuk mempermudah imajinasi kita dalam menggambarkan hirarki dari file VRML dan jangan diterima secara mati. Node yang berisi field inipun bisa juga berisi node lainnya, sehingga dapat membentuk struktur kaskading. Untuk jelasnya baca bagian berikut yang menjelaskan tentang node ini.

Untuk bisa membayangkan bagaimana kira-kira struktur dari VRML ini, baiklah kita lihat contoh sederhana berikut.

```
#VRML V2.0 utf8
Group {
   children [
     Shape{
       appearance Appearance {
          material Material {
               diffuseColor 1 0 0
          }
       }
     geometry Box{size 2 2 2}
     }
   ]
}
```

Contoh di atas menggambarkan scene yang terdiri dari environment (dengan nilai default, yaitu latar belakang warna gelap) dan sebuah kotak berwarna merah yang berukuran *2 x 2 x 2* (lihat Gambar 3).



Gb 3. Scene terdiri dari kotak berwarna merah dengan latar belakang gelap

Sekedar keterangan singkat dari source code di atas, `Group` adalah salah satu jenis node yang mempunyai field `children`. Dalam field `children` didefinisikan lagi node `Shape` yang mempunyai field `appearance` dan `geometry`. Dalam field `geometry` didefinisikan node `Box` yang mempunyai field `size`, yang adalah informasi geometri dari objek kotak. Kalau Anda perhatian, node selalu diawali dengan huruf besar, sedang field dengan huruf kecil.

Sekarang mari kita ikuti pembahasan yang lebih ditel dari struktur yang membentuk file VRML.

Header

Header digunakan untuk mengidentifikasi jenis encoding yang dipakai dalam membangun file VRML. Format header ini seperti berikut:

```
#VRML V2.0 <encoding type> [optional comment] <line terminator>
```

Pada versi VRML97 dalam praktiknya hanya digunakan tipe encoding `utf8`, sedang pada versi 1.0 dipakai `ascii`. Baris header ini harus ditulis pada baris pertama dari file VRML. Baris berikutnya yang berawalan `#` akan diinterpretasi sebagai kommentar. Untuk diingat, VRML ini mengenal perbedaan antara huruf besar dan kecil. Contoh baris header adalah berikut:

```
#VRML V2.0 utf8 Tutorial mwmag
#Ini adalah baris komentar
```

Node

Node dapat dibayangkan sebagai jenis objek dasar untuk membangun sebuah scene. Sedang kumpulan dari scene-scene ini nantinya akan membentuk dunia virtual. Dengan kata lain, node adalah objek dasar yang diperlukan untuk merealisasikan dunia virtual. Contoh tipe dari objek dasar ini adalah bentuk geometri, tampilan permukaan, latar belakang, sensor dan sebagainya. Dalam VRML terdapat lebih dari 50 tipe node standar yang sudah didefinisikan (builtin). Untuk mengklasifikasikan node ini, tentu saja bisa bermacam-macam menurut

kategori yang kita inginkan. Agar tidak bingung, kita akan mengklasifikannya menurut fungsi.

- Grouping. Dengan node tipe ini node-node lainnya dapat dikelompokkan menjadi satu grup. Harus diperhatikan, tidak semua node dapat digunakan di sini. Beberapa node penting yang termasuk jenis ini antara lain: `Transform` yang sistem koordinatnya akan dijadikan referensi bagi node-node di dalamnya, node `Collision`, `Group` dan lain-lain.
- Geometri dan penampakan. Dengan node ini maka orang dapat membangun berbagai jenis objek geometri, seperti kubus, silinder, bola, kerucut dan sebagainya. Objek geometri ini dapat mempunyai penampakan, seperti warna, jenis materi, textur yang ditentukan dengan bantuan node jenis ini. Node `Shape`, `Appearance`, `Material` termasuk node jenis ini.

Tipe node yang lain akan dibicarakan dalam artikel berikutnya, agar pembaca dapat lebih berkonsentrasi dengan tipe di atas dahulu.

Untuk menggunakan node, harus diperhatikan cara menulis sintaksnya yang didefinisikan demikian:

```
[DEF <name>] <nodeType> {
  <body>
}
```

Kolom `[DEF <name>]` tidak harus digunakan, dan hanya dipakai jika scene yang akan kita bangun dengan node ini ingin digunakan lagi, dengan cara memberinya sebuah nama sebagai identifikasi dari objek yang bersangkutan. Kolom `<nodeType>` adalah tipe node, yang selalu berawalan dengan huruf besar. Sedang `<body>` berisikan field yang akan menentukan sifat dan perilaku dari node yang bersangkutan.

Agar tidak jenuh dengan teori, kita akan membuat scene yang memuat sebuah objek silinder. Untuk itu kita menggunakan node `Shape`.

```
Shape {
   appearance Appearance {
      material Material {
      }
   }
   geometry Cylinder {
   }
}
```

Scene di atas tidak mempunyai nama, dan nilai field dari node `Material` maupun node `Cylinder` menggunakan nilai default VRML. Di VRML, field dari kebanyakan node mempunyai nilai default seandainya field yang bersangkutan tidak didefinisikan. Kita akan membahas lebih ditel lagi tentang node `Shape` dan `Transform` di bagian selanjutnya pada artikel ini. Tetapi terlebih dulu kita akan belajar tentang sifat dari field.

Field

Seperti disinggung sebelumnya, field akan menentukan sifat dan perilaku dari node yang memilikinya (parent node). Pada bahasa pemrograman, seandainya node diibaratkan sebagai struktur data, maka field bisa kita bayangkan sebagai komponen variabelnya. Namun di sini untuk mendefinisikan field, selain tipe data masih dibutuhkan kategori dari field. Secara ringkas, field di VRML terdiri dari kategori, tipe data, nama variabel dan nilainya. Sintaksnya dapat dituliskan seperti berikut:

```
<kategori> <tipe data> <nama variabel> <nilai variabel>
```

Kolom kategori pada field ini akan menentukan jenis fungsi dari variabel yang bersangkutan. Kita akan membahasnya pada tema even nanti. VRML mengenal 4 kategori dari field, yaitu `eventIn`, `eventOut`, `exposedField` dan `field`. Tiap kategori field ini mempunyai beberapa tipe data yang sama, seperti contohnya `MFInt32`, `SFNode`, `SFBool`, `SFVec3f` dan sebagainya. Nama dari tipe data pada VRML selalu mempunyai prefiks SF, yang merupakan kepanjangan dari Single Field, atau MF, yang adalah kepanjangan dari Multi Field. Bagian selanjutnya dari nama tipe data ini akan menerangkan jenis tipe data yang sesungguhnya, seperti `Bool` yang hanya bisa bernilai `TRUE` atau `FALSE`, lalu `Int32` yang merupakan bilangan desimal 32 bit dan sebagainya. Penting di sini adalah tipe data `Node` yang bisa mempunyai nilai berupa node juga, yang merupakan kunci dalam pembuatan sebuah scene.

Untuk sementara kita cukup mengenal kategori field saja dulu, yang sebenarnya dipakai hanya untuk membedakannya dengan kategori yang lain, dan tidak mempunyai kegunaan khusus. Sebaliknya kategori `eventIn`, `eventOut` dan `exposedField` mempunyai fungsi dalam berkomunikasi baik antar node maupun antara user dengan dunia virtual yang akan dibahas pada bagian tentang even di artikel kedua nanti.

Objek Geometri Dasar

Setelah mengenal struktur yang dipakai oleh VRML, tiba waktunya sekarang untuk membahas cara membuat objek 3D. Mungkin ini yang Anda nanti-nantikan. Untuk itu Anda perlu tahu node `Shape`, yaitu node yang bertanggung jawab dalam pembuatan objek dengan tampakannya. Hal ini jelas kalau kita melihat struktur node ini yang hanya mempunyai dua field:

```
Shape                                                              {
  exposedField              SFNode              appearance          NULL
  exposedField         SFNode         geometry                     NULL
}
```

Untuk sementara, kita tidak usah terpana dengan kategorinya, penting adalah variabel `appearance` dan `geometry` dengan nilai defaultnya *NULL*. Variabel `geometry` digunakan untuk mendefinisikan jenis objek 3D yang ingin Anda buat, sedang `appearance` akan menentukan tampakan dari objek 3D yang Anda buat. Secara garis besar, VRML mengenal dua jenis objek 3D, yaitu objek geometri dasar untuk membuat objek yang simpel dan objek geometri lanjutan untuk membuat objek 3D yang lebih kompleks. Di sini hanya akan dibahas objek geometri yang dasar, dengan beranggapan, setelah Anda tahu cara membuat objek yang simpel, maka untuk membangun yang lebih kompleks hanyalah masalah kemauan untuk membaca manualnya.

Objek geometri dasar terdiri dari node `Box`, `Cone`, `Cylinder`, `Sphere` dan `Text`. Sesuai dengan namanya, node ini dipakai untuk membuat objek kotak, kerucut, silinder, bola dan text. Field yang dimiliki oleh node ini tentu saja tidak sama, karena untuk membuat objek yang berbeda ini harus digunakan parameter yang berlainan. Namun yang harus diperhatikan di sini adalah posisi titik origin dari sistem koordinat objek. Ya, setiap objek yang dibuat di VRML selalu mempunyai sistem koordinat yang dijadikan referensi setiap melakukan besaran gerak. Keterangan ditel cara kerjanya akan dibahas dalam bagian Transformasi.

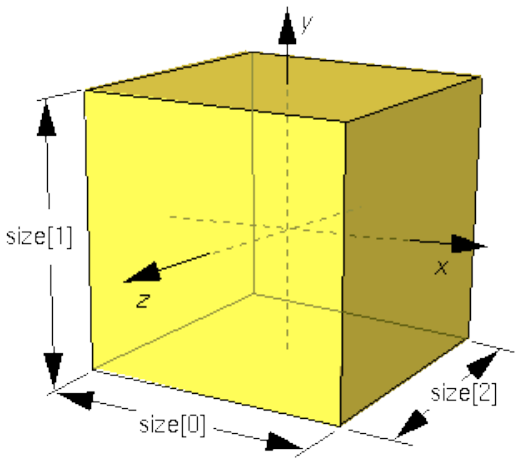
Untuk objek geometri dasar, kita hanya akan membahas node `Box` dan `Cylinder` sebagai contoh. Cara membuat objek geometri dasar lainnya dapat dengan membaca spesifikasi VRML97.

Membuat Kotak

Node `Box` memungkinkan kita untuk membuat kotak dengan panjang, lebar dan tinggi menurut keinginan kita. Node ini hanya mempunyai satu field berupa vektor yang menentukan ukuran dari kotak ini.

```
Box {
  field  SFVec3f size  2 2 2
}
```

Nilai dari variabel `size` di atas adalah nilai default jika tidak didefinisikan. Seperti terlihat pada Gambar 4, posisi titik origin dari sistem koordinat berada di pusat dari box, yaitu (0,0,0) dari sistem koordinat lokal. Sedang elemen pertama dari ukurannya adalah panjang pada sumbu *x*, elemen kedua panjang pada sumbu *y* dan elemen ketiga panjang pada sumbu *z*.

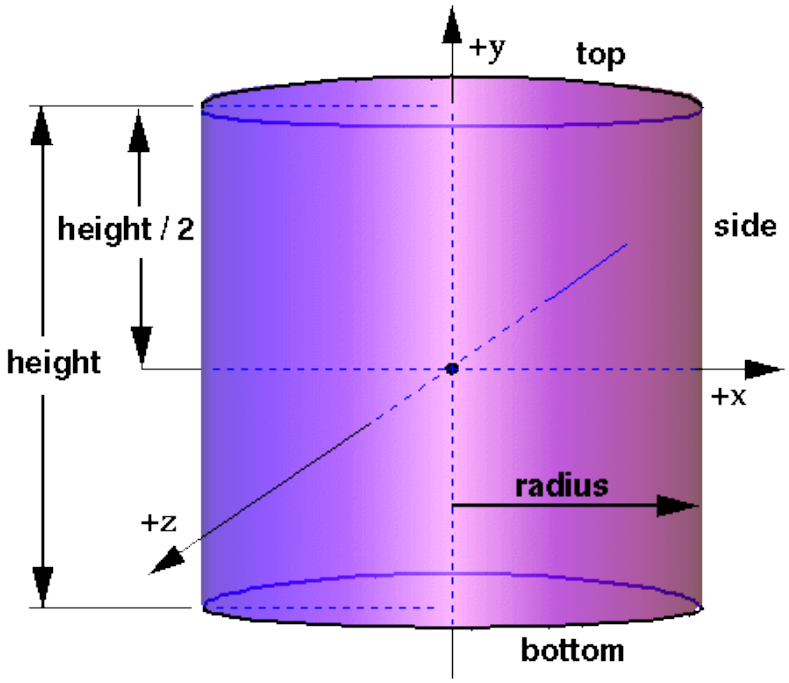


Gb 4. Node Box dengan ukuran dan sistem koordinatnya

Membuat Silinder

Node `Cylinder` dipakai untuk membuat objek silinder. Selain mempunyai field `height` dan `radius` sebagai parameter objek, node ini mempunyai 3 field lainnya masing-masing dari tipe `SFBool`, yaitu `top`, `bottom` dan `side` yang mengindikasikan apakah sisi yang disebut dalam field nyata atau tidak. Jika nilai ini `FALSE` maka pada sisi yang bersangkutan tidak akan

dilakukan proses rendering dan sisi ini tidak dapat dipakai dalam proses deteksi kolisi (collision detection). Letak sistem koordinat dan parameter geometri dari objek ini ditampilkan pada Gambar 5.



Gb 5. Node Cylinder dengan ukuran dan sistem koordinatnya

Struktur dari node ini adalah seperti berikut.

```
Cylinder {
  field    SFBool    bottom  TRUE
  field    SFFloat   height  2
  field    SFFloat   radius  1
  field    SFBool    side    TRUE
  field    SFBool    top     TRUE
}
```

Bagaimana Hasil Tampilannya?

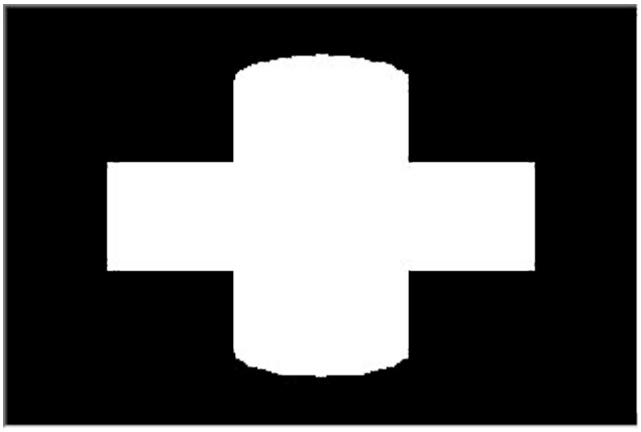
Setelah berasyik-asyik dengan teori, tiba saatnya kita membuat objek 3D pertama yang kali ini terdiri dari kotak dan silinder. Kita akan membuat kotak dengan ukuran *4 x 1 x 2* dan silinder dengan tinggi 3 dan diameter 1, tentu saja satuannya adalah *meter*. Source code VRML untuk itu adalah seperti berikut:

```
#VRML V2.0 utf8 Tutorial mwmag
Shape{
  geometry Box {
     size 4 1 2
  }
}

Shape{
  geometry Cylinder {
     height  3
     radius  1
```

```
  }
}
```

Kalau file VRML dengan source code seperti tertulis di atas kita load ke browser, maka kita akan lihat tampilannya seperti pada Gambar 6. Ada yang merusak mata? Yah, kotak dan silinder yang kita buat saling tumpang tindih, dan mempunyai warna yang sama, terus kok masih kelihatan 2D ya? No problem, bagian berikut akan mengulas cara memberi warna permukaan dan cara menentukan posisi dari objek. Masalah 2D, saya sengaja tidak mempergunakan navigasi dari browser, sehingga navigasinya menggunakan nilai default dari file VRML. Dan karena kita belum definisikan, makanya masih kelihatan 2D. Lalu bagaimana cara memberi nilai untuk navigasi ini?



Gb 6. Objek kotak dan silinder tanpa navigasi

Sistem Navigasi

Adalah jelas, bahwa lebih mudah menggunakan navigasi dari browser untuk berjalan-jalan di dunia virtual. Kita tinggal klik mouse, dan memilih jenis navigasi yang kita inginkan. Lalu dengan menggeser-geser mouse, kita dapat mengeset nilai variabel dari jenis navigasi yang telah kita pilih.

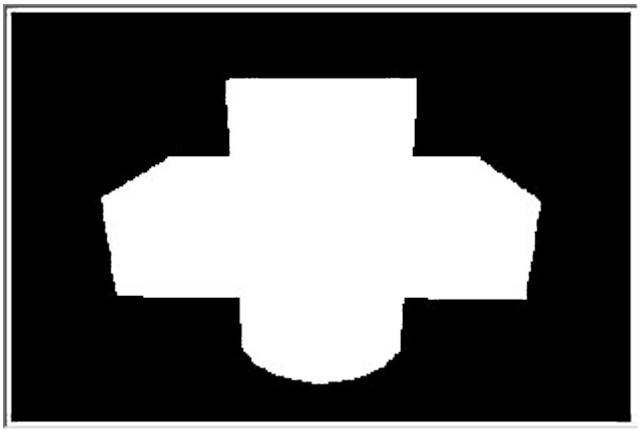
Namun, walau kita tak bisa lepas dari navigasi lewat browser ini—mungkin hanya untuk sekedar diketahui—kita juga bisa memilih jenis navigasi dan mengeset nilai variabelnya lewat node. Untuk keperluan ini digunakan node `NavigationInfo` dan `Viewpoint`. Kita tidak akan membahas lebih ditel tentang ini. Hanya dengan melihat field yang ada pada kedua node etrsebut, kita bisa mengkira-kira fungsi dari field yang bersangkutan.

```
NavigationInfo {
  eventIn      SFBool   set_bind
  exposedField MFFloat  avatarSize      [0.25, 1.6, 0.75]
  exposedField SFBool   headlight       TRUE
  exposedField SFFloat  speed           1.0
  exposedField MFString type            ["WALK", "ANY"]
  exposedField SFFloat  visibilityLimit 0.0
  eventOut     SFBool   isBound
}

Viewpoint {
  eventIn      SFBool     set_bind
  exposedField SFFloat    fieldOfView    0.785398
  exposedField SFBool     jump           TRUE
```

```
  exposedField SFRotation orientation     0 0 1 0
  exposedField SFVec3f    position        0 0 10
  field        SFString   description     ""
  eventOut     SFTime     bindTime
  eventOut     SFBool     isBound
}
```

Sekali lagi, abaikan dulu kategori dari field di atas. Jenis navigasi ditentukan oleh variabel `type` pada node `NavigationInfo`. Tipe navigasi ini ada beberapa macam, seperti "ANY", "WALK", "EXAMINE", "FLY", dan "NONE". Sedang untuk mendapat kesan “melihat 3D”, nilai dari `orientation` dan `position` dari node `Viewpoint` harus diset. Karena kita belum membahas tentang transformasi dari sistem koordinat, untuk sementara sekedar tahu saja sudah cukup. Gambar 7 adalah hasil setelah menggunakan navigasi browser dan memilih tipe "EXAMINE" dan mengeset nilai `orientation` dan `position` dengan cara menggeser posisi mouse.



Gb 7. Objek kotak dan silinder dengan navigasi

Memberi Tampakan Muka

Oke, sampai sekarang kita telah dapat membuat dua macam objek 3D: kotak, dan silinder. Namun tampakan permukaan dari semua objek ini masih sama, yaitu warna permukaan objek selalu putih dengan latar belakang hitam, Nilai warna ini memang nilai default dari VRML. Untuk memberikan tampakan muka dari objek, kita harus memakai node `Appearance`, yang tentu saja hanya dapat digunakan dalam node `Shape`. Node ini mempunyai 3 field dari tipe data `SFNode` yaitu, `material`, `texture` dan `textureTransform` yang strukturnya adalah seperti berikut:

```
Appearance {
  exposedField SFNode material          NULL
  exposedField SFNode texture           NULL
  exposedField SFNode textureTransform  NULL
}
```

Seperti namanya, dengan node `Material` kita tidak hanya dapat memberikan warna muka, melainkan juga properti lain yang dimiliki oleh sebuah benda, seperti tingkat transparensi, tingkat refleksi dan sebagainya. Tentang node `Material` ini kita akan membahas lebih ditel lagi. Node ini mempunyai 6 field yang strukturnya demikian:

```
Material {
  exposedField SFFloat ambientIntensity  0.2
  exposedField SFColor diffuseColor      0.8 0.8 0.8
  exposedField SFColor emissiveColor     0 0 0
  exposedField SFFloat shininess         0.2
  exposedField SFColor specularColor     0 0 0
  exposedField SFFloat transparency      0
}
```

Berikut adalah keterangan dari masing-masing field:

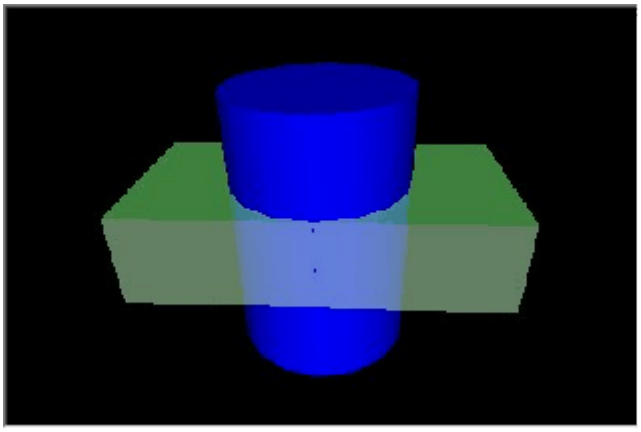
- `ambientIntensity` adalah jumlah intensitas cahaya lingkungan yang dipantulkan oleh objek yang bersangkutan;
- `diffuseColor` adalah warna normal dari objek;
- `emissiveColor` adalah field yang digunakan untuk melakukan pemodelan terhadap sifat "mengkilat" dari objek yang bersangkutan;
- `shininess` adalah tingkat refleksi dari objek;
- `specularColor` adalah warna highlight yang digunakan pada field `shininess`.

Seperti terlihat, 3 variabel dari field di atas mempunyai tipe data `SFColor` yang digunakan untuk menggambarkan warna yang diinginkan. Di VRML tipe data ini memuat tiga variabel, masing-masing adalah variabel untuk warna merah (R), warna hijau (G) dan warna biru (B). Nilai dari warna ini diskalakan antara 0 dan 1, di mana nilai 0 berarti hitam, dan 1 adalah warna penuh yaitu putih.

Sekarang kita akan membuat objek kotak dan silinder yang telah kita buat sebelumnya agar menjadi tampak lebih hidup. Untuk itu kita akan mengeset properti material dari kedua objek tersebut. Agar kedua objek tersebut kelihatan, kita akan membuat objek kotak menjadi transparan dan mempunyai sifat "berkilap" dengan warna 0 0,8 0. Sedang objek silinder hanya akan kita kasih warna biru (0 0 1). Source code untuk itu adalah seperti berikut:

```
#VRML V2.0 utf8 Tutorial mwmag
Shape{
  appearance Appearance {
     material Material {
       emissiveColor 0 0.8 0
       transparency  0.5
    }
  }
  geometry Box {
     size 4 1 2
  }
}
Shape{
  appearance Appearance {
     material Material {
       diffuseColor  0 0 1
    }
  }
  geometry Cylinder {
     height  3
     radius  1
  }
}
```

Tampilan dari source code di atas dapat dilihat pada Gambar 8.



Gb 8. Objek kotak yang transparen dan "berkilap" dengan silinder warna biru

Untuk mendapatkan tampilan muka dari jenis material tertentu yang diinginkan, tentu saja dibutuhkan pengalaman, selain sedikit perhitungan, untuk mencoba-coba nilai dari variabel di atas. Tentu saja "bakat" sebagai seorang berjiwa seni juga akan membantu mendapatkan hasil yang optimal, baik dari waktu yang digunakan maupun hasilnya.

Bagaimana Dengan Latar Belakang?

Apakah Anda memperhatikan, bahwa scene yang kita bikin tampak gelap? Benar, ini diakibatkan warna latar belakangnya yang hitam. Kalau Anda menginginkan warna yang lain, itu bukan masalah di VRML. Bahkan gambar yang Anda sukai juga dapat dijadikan sebagai latar belakang. Untuk tujuan tersebut kita gunakan node `Background`. Untuk memakai warna sebagai latar belakang, ada 4 field yang dapat diset agar mendapatkan warna yang kita kehendaki.

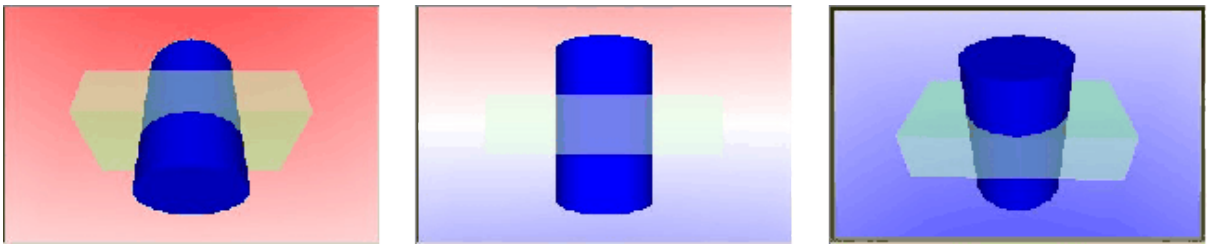
```
Background{
  exposedField MFFloat  groundAngle  []
  exposedField MFColor  groundColor  []
  exposedField MFFloat  skyAngle     []
  exposedField MFColor  skyColor     0 0 0
}
```

VRML menggunakan langit dan ground sebagai objek geometri untuk melakukan pemodelan terhadap latar belakang di dunia virtual. Langit (sky) sebagai latar belakang di VRML didefinisikan sebagai bola yang diameternya tak terhingga panjangnya yang mengelilingi dunia virtual kita. Dengan field `skyColor` warna langit ini bisa kita set. Kalau kita menginginkan warna yang terdegradasi kita dapat memanfaatkan field `skyAngle`. Harap diperhatikan, bahwa field `skyColor` dan `skyAngle` adalah variabel vektor yang bisa mempunyai beberapa nilai. Untuk membuat efek degradasi, banyak elemen warna yang dibutuhkan pada field `skyColor` harus 1 lebih banyak dari banyak elemen yang didefinisikan pada field `skyAngle`. Warna pada elemen pertama dari `skyColor` adalah warna pada posisi kutub atas dari langit. Posisi berikutnya, diukur dari posisi ini ke arah bawah sebesar sudut yang ditentukan oleh nilai pada elemen pertama dari field `skyAngle`, akan diberi warna pada elemen kedua dari `skyColor`. Warna di antara posisi pertama dan kedua ini adalah hasil degradasi dari warna pada kedua posisi tersebut. Untuk membuat efek degradasi antara posisi kedua dengan berikutnya kita tinggal mendefinisikan warna dan posisi berikutnya pada kedua

field `skyColor` dan `skyAngle`. Bingung? Oke, kita akan terangkan dengan memakai bantuan tabel dan gambar. Misalnya kita definisikan kedua field tersebut seperti berikut:

```
Background {
  skyColor [
     1  0  0
     1  1  1
     0  0  1
  ]
  skyAngle [1.57 3.14]
}
```

Ini dapat kita tuliskan dalam bentuk tabel seperti terlihat pada Tabel 2. Lebih jelas bukan? Pada posisi kutub atas warnanya adalah 1 0 0, posisi 1,57 radian berwarna 1 1 1, posisi 3,14 berwarna 0 0 1. Warna di posisi antara kutub atas dan 1,57 radian hasil degradasi antara warna 1 0 0 dengan warna 1 1 1. Warna di posisi antara 1,57 radian dan 3,14 radian adalah hasil degradasi antara warna 1 1 1 dan 0 0 1. Hal ini bisa kita lihat hasilnya pada Gambar 9.



Gb 9. Warna background dengan efek degradasi, (i) kutub atas dari langit; (ii) horizon; (iii) kutub bawah langit

Seperti langit, ground juga didefinisikan sebagai bola dengan diameter yang tak terhingga, namun letaknya masih berada di dalam bola langit. Beda bola ground dengan bola langit ini, jika warna ground tidak didefinisikan, maka bola gound ini menjadi transparen, sehingga latar belakang dari bola langit menjadi kelihatan. Selain itu, seandainyapun warna bola ground ini didefinisikan, hanya posisi yang warnanya didefinisikan akan kelihatan, posisi yang warnanya tidak didefinisikan akan menjadi transparen. Untuk memberi warna pada bola ground, caranya seperti pada bola langit, hanya bedanya letak posisi pertama dari ground berada di kutub bawah dari bola ground ini dan sudut dihitung dari bawah ke atas.

Nah, lalu kalau kita mau menjadikan image sebagai latar belakang bagaimana? Untuk itu digunakan field yang lain seperti berikut:

```
Background {
  exposedField MFString backUrl      []
  exposedField MFString bottomUrl    []
  exposedField MFString frontUrl     []
  exposedField MFString leftUrl      []
  exposedField MFString rightUrl     []
  exposedField MFString topUrl       []
}
```

Untuk penempatan image, dibuat konsep kubus yang besarnya tak terhingga dan berada di dalam bola langit dan bola ground. Sehingga ada 6 sisi, yaitu sisi atas, bawah, sisi depan, belakang, dan sisi kiri, kanan, untuk menempatkan image kita. Disarankan, untuk dapat

melihat latar belakang langit dan ground (seandainya ada), image yang dipakai juga punya latar belakang transparan.

Nah, sekarang kita dapat menempelkan image kita pada salah satu sisi dari kubus seperti yang kita inginkan. Caranya tinggal mengeset nama file dari image kita ke dalam variabel dari field di atas. Nama file ini mempunyai format URL, sehingga tidak harus berada di lokal.

Transformasi Dari Sistem Koordinat

Di Jerman ada pepatah yang mengatakan, "Ende gut alles gut", yang kurang lebih artinya "kalau akhirnya bagus, maka semua menjadi bagus". Di bagian terakhir dari artikel pertama ini kita akan membicarakan tentang transformasi dari sistem koordinat. Lalu apa hubungannya dengan pepatah di atas? Penulis berharap, bahwa Anda juga bisa menyelesaikan bagian terakhir yang relatif lebih rumit dibanding bagian sebelumnya. Tidak hanya menyelesaikan tapi juga mengerti. Kalau keinginan ini terkabul, maka semua menjadi bagus.

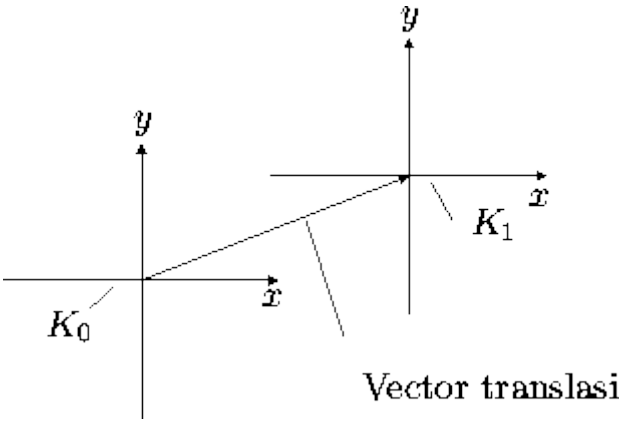
Oke, kita mulai saja. Kita kembali ke objek yang telah kita buat sebelumnya. Sebagai contoh kita sekarang ingin agar posisi dari silinder berada di sebelah kanan dari kotak. Apa yang musti dikerjakan? Jawabnya adalah node `Transform`. Node `Transform` ini mencakup proses penskalaan (scaling), rotasi dan translasi. Sifat penting yang harus diketahui adalah proses transformasi ini bersifat **akumulatif**, artinya transformasi yang sekarang ini bereferensi kepada transformasi sebelumnya. Dengan kata lain, hasil akhir transformasi adalah hasil penjumlahan dari transformasi-transformasi sebelumnya. Baiklah kita lihat dulu struktur dari node ini.

```
Transform {
  eventIn      MFNode      addChildren
  eventIn      MFNode      removeChildren
  exposedField SFVec3f     center           0 0 0
  exposedField MFNode      children         []
  exposedField SFRotation  rotation         0 0 1 0
  exposedField SFVec3f     scale            1 1 1
  exposedField SFRotation  scaleOrientation 0 0 1 0
  exposedField SFVec3f     translation      0 0 0
  field        SFVec3f     bboxCenter       0 0 0
  field        SFVec3f     bboxSize         -1 -1 -1
}
```

Jangan terkejut dengan jumlah field yang relatif banyak. Untuk sementara yang perlu diperhatikan adalah field `center`, `rotation`, `scale`, `scaleOrientation` dan `translation`, yang akan kita bahas di sini dan memegang peranan penting. Pada node `Transform` ini, urutan pengerjaan tidak bergantung dari urutan field saat didefinisikan, melainkan sudah fix, yaitu mula-mula proses penskalaan, rotasi, dan baru kemudian translasi. Field yang bertanggung jawab untuk itu adalah `scale`, `rotation` dan `translation`. Namun perlu diperhatikan, ketiga proses tersebut selalu berreferensi pada nilai dari field `center`. Bahkan pada proses penskalaan, sebelumnya masih harus melihat nilai dari `scaleOrientation`. Kita mulai saja dari yang mudah dulu yaitu translasi.

Translasi

Translasi memungkinkan kita untuk menempatkan objek pada posisi tertentu di dunia virtual. Sebagai referensi pengukuran digunakan titik origin dari sistem koordinat objek, seandainya nilai `center` pada node `Transform` tidak didefinisikan. Seandainya nilai `center` ini bukan `0 0 0`, maka titik `center` digunakan sebagai titik referensi yang baru pada pengukuran posisi. Untuk proses translasi dan rotasi, agar tidak membingungkan, kita tidak akan menggunakan field `center` dulu. Hal ini akan dijelaskan pada bagian penskalaan. Oke, kembali ke translasi, Gambar 10 mengilustrasikan proses translasi ini.



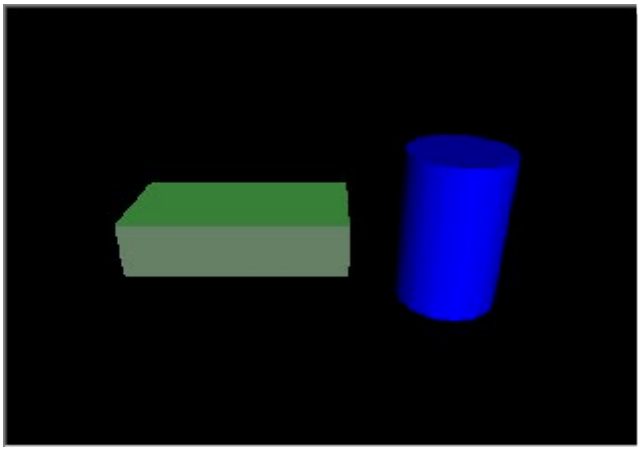
Gb 10. Proses translasi dari sistem koordinat K0 ke sistem koordinat K1

Sistem koordinat K0 adalah sistem koordinat di luar node `Transform`. Sedang sistem koordinat K1 adalah sistem koordinat di dalam node `Transform` yang menjadi sistem koordinat lokal yang baru bagi objek yang didefinisikan di dalam node `Transform`. Untuk memindahkan objek silinder kita ke sebelah kanan, maka kita dapat lakukan dengan memakai field `translation` ini dengan cara mengeset nilai *x*-nya. Kita lihat source code di bawah ini:

```
...
Transform {
  translation 4 0 0
  children [
    Shape{
      appearance Appearance {
         material Material {
           diffuseColor  0 0 1
         }
      }
      geometry Cylinder {
        height  3
        radius  1
      }
    }
  ]
}
```

Agar tidak memakan tempat, source untuk membuat objek kotak tidak ditampilkan lagi, karena sama dengan sebelumnya. Scene untuk membuat silinder sekarang berada di dalam node `Transform` yang sistem koordinatnya dipindah sebesar 4 meter ke kanan ke arah positif dari sumbu *x*. Tepatnya node `Shape` untuk objek silinder harus ditulis di dalam field

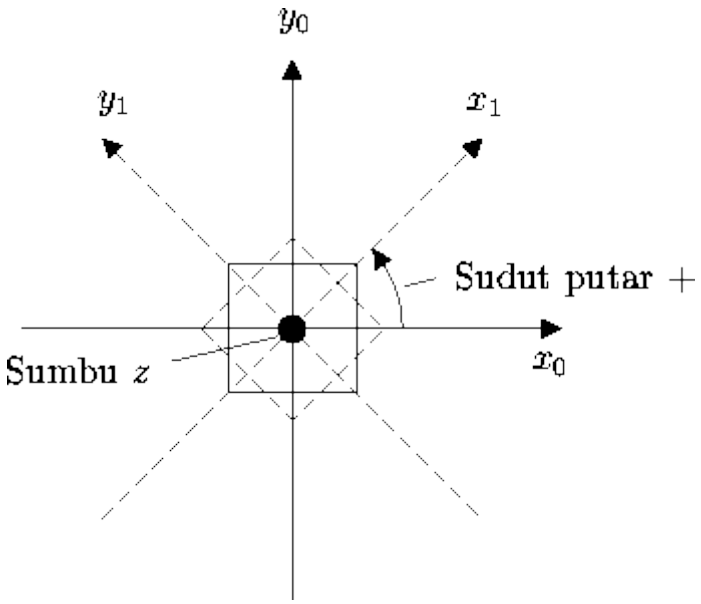
`children` dari node `Transform`. Hasil tampilan dari program kita yang baru dapat dilihat di Gambar 11.



Gb 11. Posisi titik origin dari objek silinder sekarang berada 4 meter di sebelah kanan titik origin dari kotak

Rotasi

Pada rotasi, sistem koordinat hanya akan diputar dengan sumbu dan nilai putar yang dapat didefinisikan pada field `rotation`. Sumbu putar ini berupa vektor yang arahnya ditentukan oleh 3 nilai pertama pada variabel `rotation`. Sedang nilai ke-4 menentukan nilai besarnya putaran. Gambar 12 mengilustrasikan proses rotasi ini.

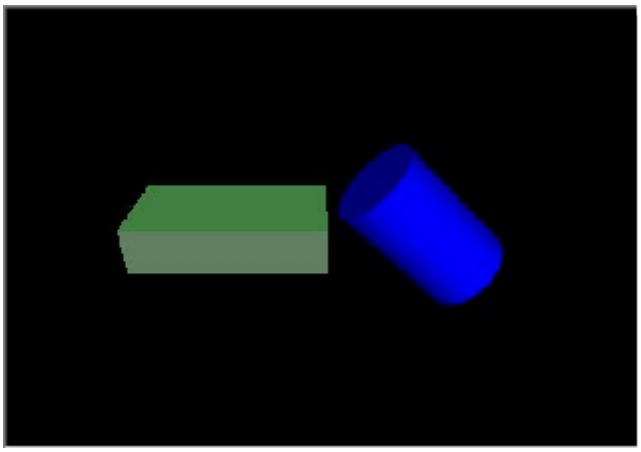


Gb 12. Proses rotasi dengan sumbu putar z dari sistem koordinat X0, Y0, Z0 ke sistem koordinat X1, Y1, Z1

Rotasi pada Gambar 12 adalah rotasi sebesar 45 derajat atau 0,785 radian dengan sumbu putar *z*, dari sistem koordinat X0, Y0, Z0 (bergaris penuh) ke sistem koordinat X1, Y1, Z1 (bergaris putus-putus). Nilai untuk `rotation` yang sepadan dengan rotasi di atas adalah 0 0 1 0.785. Rotasi ini akan kita terapkan pada objek silinder. Untuk itu kita tinggal menambah satu baris di dalam node `Transform` seperti berikut:

```
...
Transform {
  translation 4 0 0
  rotation 0 0 1 0.785
  children [
...
```

Seperti yang diharapkan, objek silinder sekarang posisinya miring diputar ke kiri sebesar 45 derajat seperti terlihat pada Gambar 13.

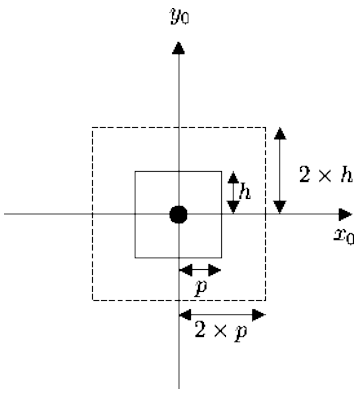


Gb 13. Posisi objek silinder sekarang terputar ke arah kiri sebesar 45 derajat

Penskalaan

Pada proses rotasi dan translasi di atas, sistem koordinat dari objek selalu dijadikan referensi, karena kita tidak mendefinisikan variabel `center`. Kita akan belajar bagaimana cara kerja variabel ini pada proses penskalaan. Hal ini tentu saja juga dapat langsung diterapkan terhadap kedua proses translasi dan rotasi.

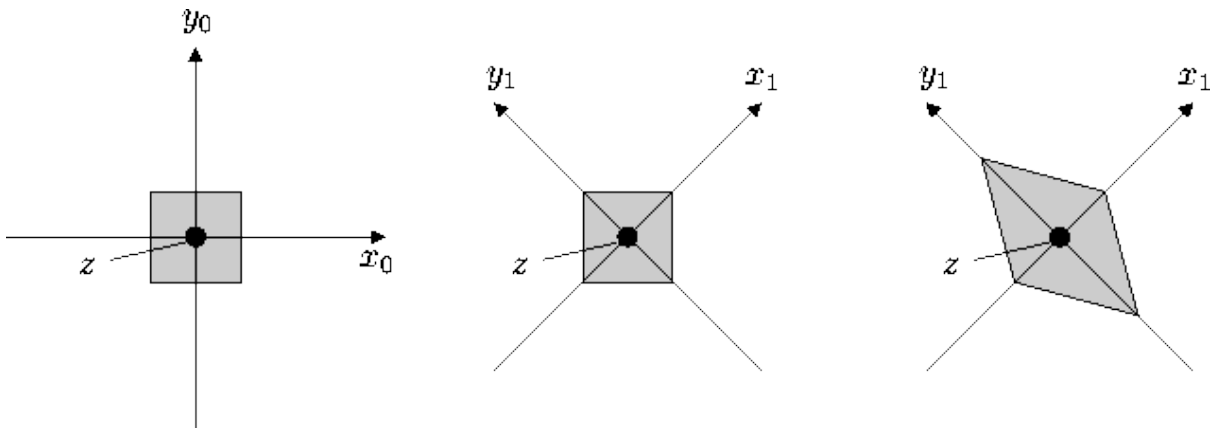
Tiga field bertanggung jawab dalam proses penskalaan, yaitu `center`, `scale` dan `scaleOrientation`. Agar mudah dimengerti, kita akan mulai dengan `scale` dulu. Dengan field ini, maka kita bisa mengubah ukuran objek geometri ke arah *x*, *y* maupun *z* dengan menuliskan baris *scale x y z*. Faktor skala ini harus positif, karena prosesnya akan diterapkan ke kedua arah dari sumbu. Nilai pada variabel `scale` akan dikalikan dengan ukuran objek yang dihitung dari titik origin objek yang bersangkutan. Gambar 14 menjelaskan proses penskalaan sepanjang sumbu *x* dan *y* masing-masing sebesar faktor 2.



Gb 14. Proses penyekalaan sepanjang sumbu x dan y masing-masing sebesar faktor 2

Seandainya pengubahan ukuran objek harus dilakukan ke arah sumbu yang lain, maka kita harus mendefinisikan sistem koordinat baru untuk dijadikan referensinya. Hal ini dapat dicapai dengan field `scaleOrientation`. Field ini mempunyai 4 variabel, 3 variabel pertama *x y z* akan membentuk vektor yang dijadikan referensi buat perputaran sistem koordinat baru. Variabel ke empat adalah besarnya sudut putar dalam radian.

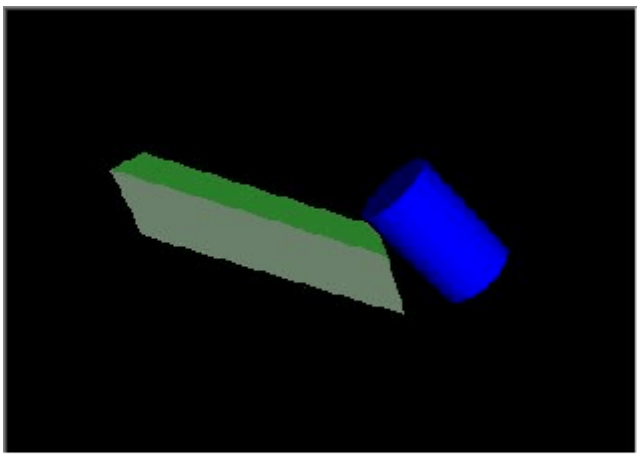
Gambar 15 memperjelas skenario ini. Kita akan melakukan penskalaan dari sebuah objek sepanjang salah satu dari sumbu diagonalnya. Oke, mula-mula kita harus membuat sistem koordinat baru yang salah satu sumbunya harus berhimpitan dengan sumbu diagonal objek yang diinginkan. Salah satu caranya adalah dengan memutar sistem koordinat objek dengan sumbu putar *z* sebesar 45 derajat. Lalu kita melakukan penskalaan, misalnya sepanjang sumbu *y* yang baru. Hasilnya terlihat pada gambar bagian ketiga. Perintah yang sepada dengan proses di atas adalah:



Gb 15. Proses penyekalaan dengan field scaleOrientation

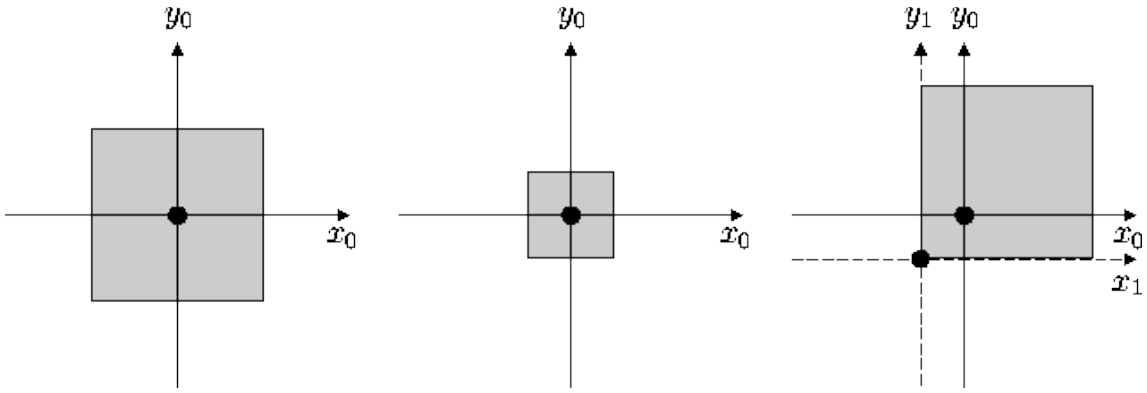
```
Transform {
   scaleOrientation 0 0 1 0.785
   scale 1 2 1
}
```

Agar lebih percaya lagi, kita akan lakukan penskalaan terhadap objek kotak sepanjang sumbu diagonal seperti ilustrasi sebelumnya dengan menambahkan baris di atas ke source kita sebelumnya. Hasilnya dapat dilihat pada Gambar 16.



Gb 16. Objek kotak diskalakan sepanjang sumbu diagonal di bidang x-y sebesar faktor 2

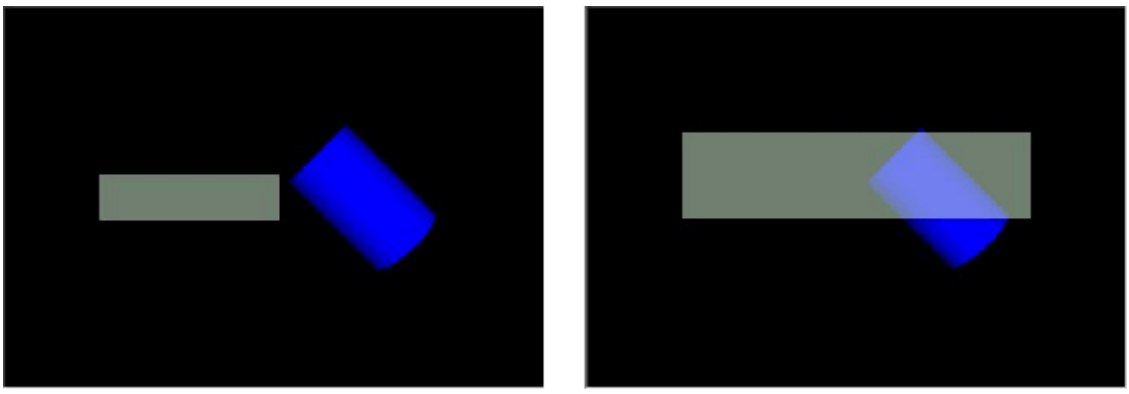
Sampai di sini proses pengubahan ukuran terjadinya selalu simetris, ke arah positif dan negatif dari sumbu yang dikehendaki. Adakalanya kita menginginkan, agar proses penskalaan ini hanya untuk satu arah saja. Ini dapat dicapai dengan field `center`. Titik `center` ini menspesifikasikan titik referensi baru (sebagai pengganti titik origin dari objek geometri yang bersangkutan) sebelum dilakukan proses dengan `scaleOrientation` dan `scale`. Dengan kata lain, titik `center` ini akan menggantikan fungsi titik origin secara temporar dalam proses translasi, rotasi maupun penskalaan. Masih bingung? Baiklah, kata orang gambar itu lebih mudah dimengerti daripada seribu kata-kata. Kita akan melihat Gambar 17 sebagai ilustrasinya.



Gb 17. Proses penskalaan dengan field center

Objek kotak yang berada di tengah adalah objek yang belum diskalakan. Sedang objek sebelah kiri adalah hasil penskalaan sepanjang sumbu *x* dan *y* sebesar faktor 2 tanpa field `center` seperti sebelumnya telah kita bahas. Penskalaan ini menghasilkan objek yang lebih besar dan simetris terhadap sumbu *x* dan *y*. Objek pada gambar sebelah kanan adalah hasil penskalaan sepanjang sumbu dan dengan faktor yang sama seperti pada gambar sebelah kiri. Hanya saja di sini kita tidak menggunakan titik origin dari sistem koordinat X0,Y0 sebagai referensinya, melainkan memakai titik origin dari sistem koordinat yang baru, yaitu X1,Y1, yang didefinisikan pada field `center` yaitu -1 -1 0.

Oke, sekarang kita akan melakukan penskalaan memakai field `center` terhadap objek kotak kita. Sebagai pembanding, gambar dari objek sebelum penskalaan juga akan ditampilkan, seperti terlihat pada Gambar 18.



Gb 18. Objek kotak diskalakan dengan titik center -2 -0.5 0 sepanjang sumbu x dan y sebesar faktor 2

Nah, Anda lihat bukan, pada objek kotak terjadi perbesaran ukuran “hanya” ke arah positif sumbu *x* dan *y*.

Penutup

Sampai di sini, objek yang kita bikin masih bersifat statik, tidak bisa berbuat apa-apa. 3D itu bagus, tapi kalau belum bisa bergerak dan bereaksi terhadap dunia luar, tetap saja masih kurang. Baiklah, pada artikel selanjutnya nanti kita akan berkenalan dengan konsep even. Konsep ini memegang peranan penting sebagai satu-satunya cara berkomunikasi antara kita dengan objek di dunia virtual, objek, dengan objek maupun objek dengan lingkungan virtualnya. Sampai nanti.