LAPORAN PEMBUATAN OBJEK

**Untuk Memenuhi Nilai Ujian Akhir Semester**

**Mata Kuliah Grafika Komputer Semester Ganjil Tahun Ajaran 2021**

**Disusun oleh :**

**Wilson Otto Sasongko - C14190017**

**Denzel Daniel D'Assante Tangsaputra - C14200160**

**Andreas Hariyanto - C14190198**

**Dosen:**

**LILIANA, S.T., M.Eng., Ph.D.**

**PROGRAM STUDI INFORMATIKA**

**FAKULTAS TEKNOLOGI INDUSTRI**

**UNIVERSITAS KRISTEN PETRA**

**SURABAYA**

# Pendahuluan

Untuk memenuhi nilai Ujian Akhir Semester mata kuliah Grafika Komputer, kelompok kami melanjutkan ide tema dari proyek Ujian Tengah Semester yaitu “*Latest Technologies from the Myssilia Clan*” (Teknologi Terbaru dari Klan *Myssilia*) dengan menambahkan fitur - fitur yang membuat lingkungan menjadi lebih kaya, seperti Lighting, Camera, dll.

# Pembagian Tugas

**Andreas:**

* Abstract Light
* Spotlight
* Camera

**Wilson:**

* Directional Light
* Point Light
* Phong Shading

**Denzel:**

* Tetrahedron & Octahedron
* Camera Choreography
* Penataan Objek dan Cahaya
* Pembuatan Normal Vector
* Blinn-Phong Shading
* Instancing
* Skybox (Cubemap)
* Video Editing

# Pembuatan Objek

Objek - objek utama yang digunakan sama persis dengan proyek Ujian Tengah Semester, dengan dua tambahan jenis objek:

1. Octahedron

Octahedron memiliki delapan sisi yang terbentuk oleh segitiga. Fungsi createOctahedron() menerima enam variabel untuk menentukan letak objek di dunia dan jarak dari titik pusat ke keenam titik di ketiga sumbu xyz. Bentuk Octahedron digunakan untuk menandakan lokasi Point Light.



1. Tetrahedron

Tetrahedron memiliki empat sisi yang terbentuk oleh segitiga. Fungsi createTetrahedron() menerima empat variabel untuk menentukan letak objek di dunia dan panjang rusuk dari Tetrahedron. Bentuk Tetrahedron digunakan untuk menandakan lokasi Spotlight.

Diketahui rusuk (a) sebuah tetrahedron, maka tinggi tetrahedron tersebut adalah:

(Köller, 2001)

Jarak antara titik tengah dan verteks tetrahedron adalah:

(Köller, 2001)

Rumus tersebut digunakan dalam pembuatan Tetrahedron:

temp\_vector.Y = MathF.Sqrt(3 / 8.0f) \* edgeLength + y\_; // line 309; spotlight.cs



Terdapat juga beberapa objek baru untuk menghias lingkungan sebagai berikut:

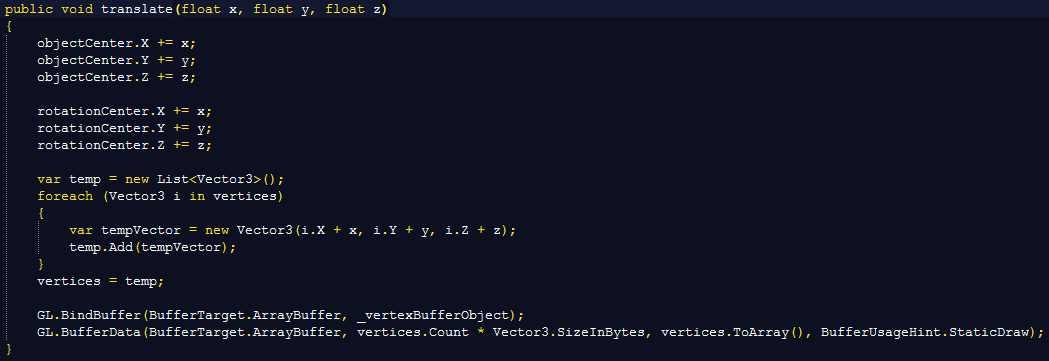
* Lantai - Cuboid
* Pedestal Spaceship - Cuboid
* Pedestal UFO - Cylinder
* 100 Asteroid Abstrak - Cuboid
* 500 Gunung Abstrak - Cuboid

# Perubahan Fungsi Transformasi

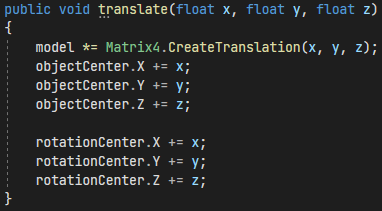
Pada proyek versi sebelumnya, fungsi transformasi menggerakkan seluruh verteks dari objek - objek yang bersangkutan secara manual, dan mengirim verteks - verteks yang baru tersebut ke *vertex buffer object*. Metode ini sangat tidak efisien karena terdapat ratusan atau ribuan verteks per objek (puluhan ribu sampai ratusan ribu verteks untuk satu proyek) yang harus diubah. Karena hal tersebut, performance dari program proyek menjadi kurang baik.

Untuk memperbaiki masalah ini, kelompok kami memutuskan untuk menggunakan matriks transformasi, sehingga jumlah operasi berkurang secara drastis pada objek - objek yang memiliki jumlah verteks yang tinggi.

Contoh untuk fungsi translate():



(Code lama, mengubah verteks secara manual dan mengirimnya ke *VBO*)



(Code baru, mengalikan *model matrix* dengan matriks translasi)

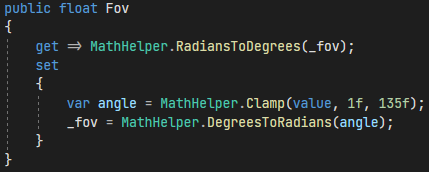
# Kamera

OpenGL/OpenTK tidak mengenal konsep kamera, sehingga untuk mensimulasi kamera, digunakan sebuah *view matrix* yang akan mengubah posisi dari seluruh objek, sehingga saat kamera digerakkan ke sebuah arah, objek - objek akan bergerak ke arah sebaliknya. (Vries, n.d.)

Untuk proyek kami, kami menggunakan class Camera.cs dari LearnOpenTK. Untuk memadai keperluan proyek, terdapat perubahan dari class Camera.cs yang original, yaitu:

* *Field of view Clamp*

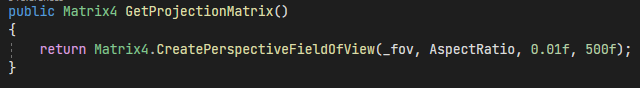
Terdapat fungsi MathHelper.Clamp() yang mengunci nilai sudut *field of view* (fov) antara 1 dan 45 derajat. Karena nilai default fov adalah 90 derajat, batas atas *clamp* diubah menjadi 135 derajat supaya fov tidak terkunci di 45 derajat saat mengubah fov.



(Camera.cs; line 79 - 87)

* *Frustum Far Plane*

Karena lingkungan proyek mencakup daerah yang amat luas, *frustum far plane* yang berjarak 100 dari posisi kamera tidak dapat menunjukkan seluruh objek. Nilai ini diubah menjadi 500 untuk mencakupi seluruh objek tanpa ada yang terpotong oleh *frustum far plane*.

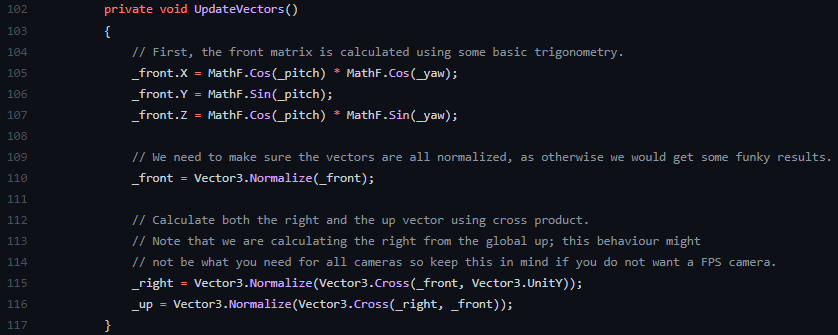


(Camera.cs; line 96 - 99)

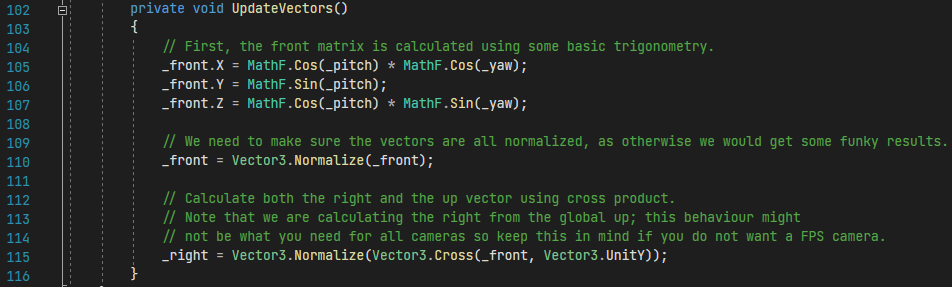
* Perhitungan *up vector*

Untuk kontrol kamera, supaya gerakan kamera lebih nyaman, kelompok kami memutuskan untuk mengunci *up vector* menjadi *unit vector* positif Y, sehingga gerakan naik turun dengan tombol *Space* dan *Left Shift* tidak akan terpengaruh oleh *pitch* dari kamera.

Pada class Camera.cs original dari LearnOpenTK, dalam fungsi UpdateVectors(), terdapat perhitungan untuk nilai *up vector* yaitu *cross product* antara *right vector* dan *front vector*. Untuk proyek kami, perhitungan tersebut dihapus untuk memastikan bahwa *up vector* selalu mengarah ke sumbu positif Y.



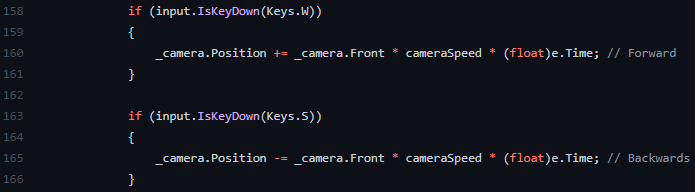
(Camera.cs dari LearnOpenTK; line 116)



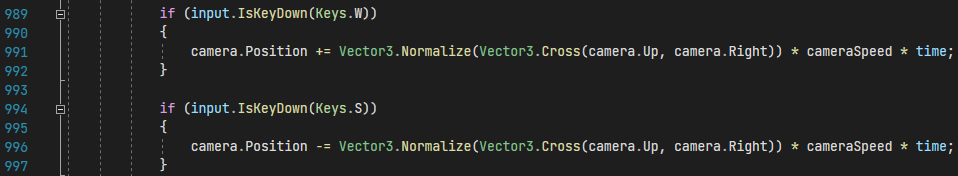
(Camera.cs; line 116 dihapus)

* Kontrol kamera

Untuk gerakan kamera, LearnOpenTK menggunakan tombol W dan S untuk maju dan mundur. Kedua tombol ini akan menggerakkan kamera dengan *front vector*. Untuk proyek kami, tombol W dan S akan menggerakkan kamera dengan *normalized cross product* antara *up vector* dan *right vector*. Perubahan ini membuat tombol W dan S untuk menggerakkan kamera tanpa mengubah posisi kamera pada sumbu Y, tetapi tetap mengikuti komponen X dan Z dari *front vector*.



(Window.cs dari LearnOpenTK; line 158 - 166)



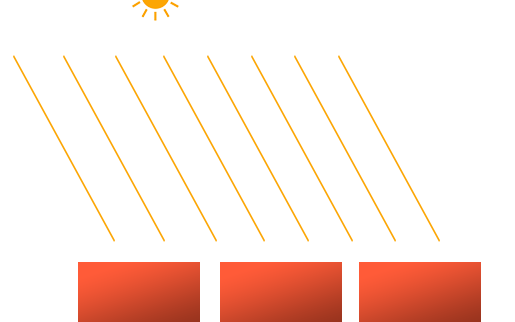
(Window.cs; line 989 - 997)

# Implementasi Cahaya

Terdapat tiga jenis cahaya yang digunakan dalam proyek ini, yaitu:

1. Directional Light

Directional Light adalah cahaya yang datang dari sebuah sumber cahaya yang terletak pada lokasi yang sangat jauh dan memiliki intensitas cahaya yang kuat, sehingga dalam proyek ini, diimplementasikan dengan memberikan shader sebuah *direction vector* yang melambangkan arah dari cahaya yang datang.



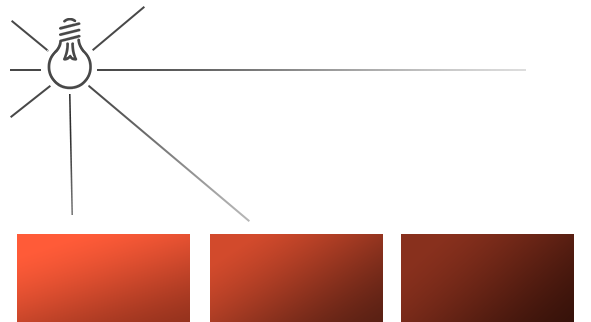
(Gambar oleh [Joey de Vries](https://learnopengl.com/Lighting/Light-casters))

Karena sumber cahaya sangat kuat, Directional Light tidak melemah saat permukaan terletak jauh dari sumber (tidak memiliki *attenuation*).

Proyek ini memiliki satu Directional Light yang memiliki warna **#1A1A3A** dan memiliki *direction vector* (-2, -4, -2). Vektor ini akan dinormalisasi sebelum dikirim ke *fragment shader*.

1. Point Light

Point Light adalah cahaya yang datang dari sebuah lokasi dan memancar ke seluruh arah. Point Light memiliki *position* yang menentukan lokasi dari Point Light tersebut.



(Gambar oleh [Joey de Vries](https://learnopengl.com/Lighting/Light-casters))

Semakin jauh sebuah cahaya dari sebuah permukaan, semakin lemah pantulan cahaya tersebut, sehingga diimplementasikan *attenuation* untuk melemahkan cahaya tergantung dari jarak antara cahaya dan permukaan.

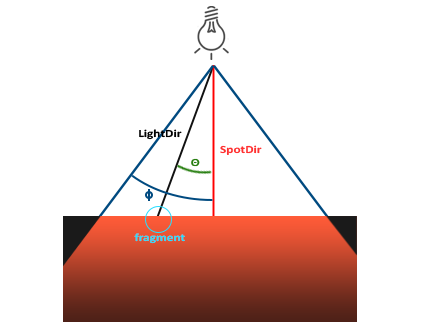
Terdapat 14 Point Light dalam proyek ini, yang terbagi sebagai berikut:

* 4 Point Light yang bergerak melingkari titik fokus dari lingkungan
* 6 Point Light yang bergerak naik turun, terletak melingkari titik fokus dari lingkungan dan mendekati pinggir dari lingkungan.
* 4 Point Light yang menempel pada podium spaceship sebelum spaceship terbang.

Seluruh Point Light berwarna putih.

1. Spotlight

Spotlight adalah cahaya yang datang dari sebuah lokasi dan memancar ke suatu arah tertentu dan memiliki sebuah sudut dimana cahaya tersebut akan melemah hingga tidak memancarkan cahaya. Spotlight memiliki vektor *position* yang menentukan lokasi, vektor *direction* yang menentukan arah pancaran cahaya, sudut *inner cutoff* yang menentukan sudut dimana cahaya mulai melemah, dan sudut *outer cutoff* yang menentukan sudut dimana cahaya menghilang secara penuh.



(Gambar oleh [Joey de Vries](https://learnopengl.com/Lighting/Light-casters))

Seperti Point Light, Spotlight juga memiliki *attenuation* untuk melemahkan cahaya tergantung dengan jarak antara cahaya dan permukaan.

Terdapat 9 Spotlight dalam proyek ini, yang terbagi sebagai berikut:

* 1 Spotlight yang mengarah kepada *Myssilia Clan Emblem* dan vektor *direction* bergerak dari kiri ke kanan selama animasi berjalan.
* 2 Spotlight yang terletak di sumbu X yang mengarah ke sisi luar dari lingkungan, dan vektor *direction* bergerak dari kiri ke kanan selama animasi berjalan.
* 3 Spotlight berwarna merah (#FF0000), hijau (#00FF00), dan biru (#0000FF) yang terletak di sisi sumbu negatif Z. Vektor *direction* dari spotlight merah dan biru bergerak dari kiri ke kanan, dan spotlight hijau bergerak melingkar selama animasi berjalan.
* 3 Spotlight yang terletak pada podium UFO mengarah ke UFO sebelum terbang.

Selain tiga spotlight yang terakhir, seluruh spotlight berwarna putih.

Untuk menentukan intensitas pantulan cahaya pada sebuah permukaan, diperlukan *normal vector* pada seluruh verteks. *Normal vector* adalah unit vektor yang tegak lurus terhadap permukaan yang bersangkutan dengan verteks tersebut. *Normal vector* adalah *direction vector*; tidak ada komponen lokasi dalam vektor tersebut.

Karena kelompok kami memutuskan untuk tidak memakai file .obj, *normal vector* harus dibuat secara matematis. Untuk menemukan *normal vector* dari berbagai objek, kelompok kami menggunakan Turunan (mata kuliah Matematika Dasar / Kalkulus I) dan alat bantu graf seperti Desmos atau GeoGebra. Berikut adalah objek - objek yang memiliki *normal vector* dan cara pembuatannya:

1. **Cuboid**

*Normal vector* dari sebuah cuboid sangat sederhana. Setiap *vertex* yang berhubungan dengan sisi tertentu akan memiliki *normal vector* searah dengan koordinat sisi tersebut. (*Vertex* di sisi kanan memiliki *normal vector* yang mengarah ke positif X, sisi kiri mengarah ke negatif X, dst.)

Untuk variasi Cuboid dalam proyek kami (Rhombic Prism & Trapezoid Prism), akan digunakan fungsi trigonometri untuk menghitung *normal vector* pada verteks yang bersangkutan pada sisi yang miring.

1. **Ellipsoid**

Sebuah Ellipsoid (dan variasi Ellipsoid, Fractioned Ellipsoid) yang memiliki radius X, Y dan Z yang sama memiliki *normal vector* yang sederhana pula. Untuk setiap verteks, vektor titik verteks dikurangi dengan titik pusat ellipsoid akan menghasilkan *direction vector* yang mengarah keluar dari titik pusat ellipsoid melalui titik verteks tersebut. Setelah dinormalisasi, vektor tersebut akan menjadi *normal vector* untuk verteks yang bersangkutan.

1. **Elliptic Cone**

Elliptic Cone memiliki dua sisi yang akan menerima *normal vector* yang berbeda, yaitu sisi samping dan sisi bawah.

*Normal vector* untuk sisi bawah sangat sederhana, yaitu *unit vector* yang mengarah ke negatif Y. *Normal vector* untuk sisi samping akan bergantung dengan gradien pada posisi verteks yang bersangkutan. Gradien didapatkan dengan , dimana:

h = tinggi elliptic cone

r = radius elliptic cone pada posisi verteks

Untuk elliptic cone yang memiliki radius X = radius Z, r = radius yang dimasukkan sebagai parameter di fungsi createEllipticCone(). Bila radius X radius Z, maka digunakanlah formula berikut untuk menghitung radius ellipse pada sudut tertentu:

(Lawrence, 1972)

a = radius X

b = radius Z

θ = sudut verteks dari sumbu X

Setelah mendapatkan gradien, *normal vector* akan dibuat dengan parameter berikut:



(Asset3d.cs; line 849)

Komponen X dari vektor dikalikan dengan cos dari sudut yang menentukan posisi verteks di sektor yang bersangkutan.

Komponen Y dari vektor adalah 1.

Komponen Z dari vektor dikalikan dengan sin dari sudut yang menentukan posisi verteks di sektor yang bersangkutan.

1. **Sinusoid Capsule**

Sinusoid Capsule memiliki *normal vector* yang cukup kompleks. Bentuk dari Sinusoid Capsule pada satu sektor adalah kurva cos.

Seperti Elliptic Cone, bila radius X radius Z, maka digunakanlah formula yang sama untuk mencari radius pada verteks.

Komponen X dari vektor adalah cos dari sudut sektor dikalikan dengan tinggi capsule dibagi dengan 𝝅.

Komponen Y dari vektor adalah radius pada verteks dikalikan dengan sin dari koordinat Y dari verteks dibagi dengan tinggi capsule dikalikan dengan 𝝅.

Komponen Z dari vektor adalah sin dari sudut sektor dikalikan dengan tinggi capsule dibagi dengan 𝝅.





(Asset3d.cs; line 923)

1. **Cylinder**

*Normal vector* dari sebuah cylinder juga cukup sederhana. Terdapat tiga jenis sisi yang memiliki *normal vector* yang berbeda, yaitu sisi atas, samping, dan bawah.

Sisi atas menggunakan unit vektor yang mengarah ke positif Y, dan sisi bawah menggunakan unit vektor yang mengarah ke dan negatif Y sebagai *normal vector*.

Sisi samping cylinder menggunakan *direction vector* yang mengarah dari titik pusat cylinder ke verteks yang bersangkutan, dengan kedua titik tersebut memiliki komponen Y yang sama.



(Asset3d.cs; line 1006)

Untuk variasi cylinder (Varied Cylinder, dimana radius sisi atas dan sisi bawah berbeda), komponen *normal vector* adalah sebagai berikut:

Komponen X adalah nilai absolut dari gradien sisi samping dikalikan dengan cos sudut sektor.

Komponen Y adalah 1 atau -1, tergantung dari kemiringan gradien.

Komponen Z adalah nilai absolut dari gradien sisi samping dikalikan dengan sin sudut sektor.



(Asset3d.cs; line 1113)

1. **Elliptic Paraboloid**

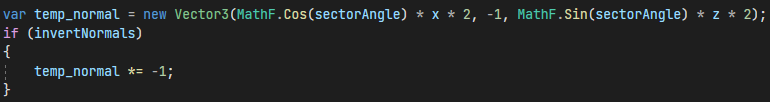
Komponen *normal vector* dari sebuah Elliptic Paraboloid adalah sebagai berikut:

Komponen X adalah 2 kali komponen X dari verteks dikalikan dengan cos sudut sektor.

Komponen Y adalah -1.

Komponen Z adalah 2 kali komponen Z dari verteks dikalikan dengan sin sudut sektor.

Karena Elliptic Paraboloid dalam proyek memiliki dua sisi, terdapat juga fungsi untuk membalik *normal vector* sehingga pantulan cahaya terlihat pada sisi yang diinginkan.



(Asset3d.cs; line 1224 - 1228)

1. **Circle**

*Normal vector* dari sebuah lingkaran adalah unit vektor yang mengarah ke positif Y. Karena penggunaan lingkaran hanya terlihat pada satu sisi, sisi satunya tidak dibuatkan *normal vector*.

1. **Octahedron & Tetrahedron**

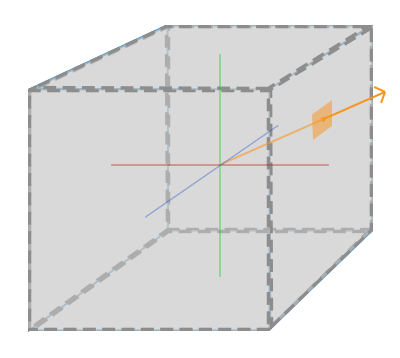
Karena kedua objek tersebut digunakan untuk representasi cahaya, kedua-duanya tidak memiliki *normal vector* dalam proyek.

Seluruh *normal vector* akan dinormalisasi sebelum dimasukkan ke List, yang kemudian akan dikirim ke *fragment shader*.

# Skybox (Cubemap)

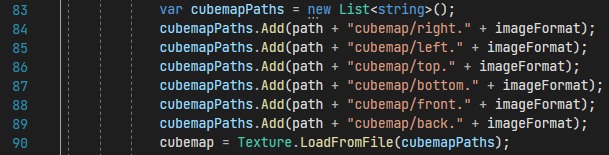
Untuk menciptakan lingkungan yang terasa luas, kelompok kami memutuskan untuk mengimplementasikan sebuah skybox menggunakan Cubemap.

Cubemap adalah sebuah texture yang memiliki enam texture dua dimensi yang membentuk sebuah kubus. Cubemap memiliki kelebihan dimana koordinat texture sebuah cubemap dapat diambil dengan sebuah *direction vector* (panjang vektor tidak berpengaruh) seperti berikut.



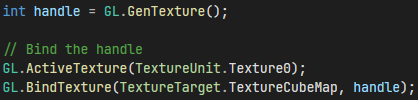
(Gambar oleh [Joey de Vries](https://learnopengl.com/Advanced-OpenGL/Cubemaps))

Untuk memasukkan path texture cubemap kedalam class Texture.cs, di buatlah *overload* dari *function* LoadFromFile() di Texture.cs dari LearnOpenTK untuk menerima sebuah List<string>.



(Window.cs; line 83 - 90)

Cubemap adalah sebuah texture, maka untuk membuat sebuah cubemap, sebuah texture akan di-*generate* dan di-*bind* pada TextureTarget.TextureCubemap.



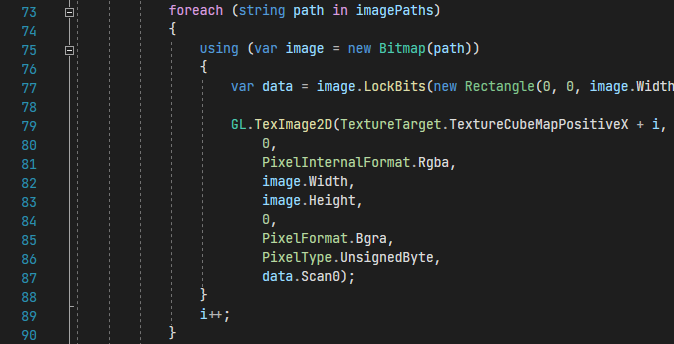
(Texture.cs; line 63 - 67)

Kemudian, karena Cubemap memiliki enam sisi, GL.TexImage2D() akan dipanggil enam kali, dan TextureTarget yang dimasukkan akan berubah sesuai dengan sisi dari Cubemap yang akan digambar.

Berikut adalah enam Texture Target yang dimiliki cubemap:

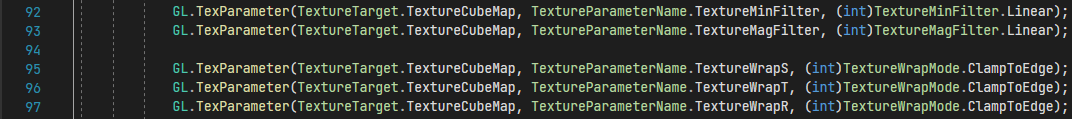
| **Texture Target** | **Orientation** |
| --- | --- |
| TextureTarget.TextureCubeMapPositiveX | Kanan |
| TextureTarget.TextureCubeMapNegativeX | Kiri |
| TextureTarget.TextureCubeMapPositiveY | Atas |
| TextureTarget.TextureCubeMapNegativeY | Bawah |
| TextureTarget.TextureCubeMapPositiveZ | Depan |
| TextureTarget.TextureCubeMapNegativeZ | Belakang |

Kelebihan dari *enum* milik OpenGL adalah nilai int dari *enum - enum* tersebut berurutan setelah yang sebelumnya, sehingga mendapatkan seluruh TextureTarget untuk cubemap dapat dilakukan dengan menggunakan for loop yang mengincrement nilai TextureTarget.TextureCubeMapPositiveX dengan satu setiap iterasi untuk mendapatkan *enum* untuk sisi lainnya:



(Texture.cs; line 73 - 90)

Setelah semua texture di-*generate*, texture akan diberi metode *wrapping* dan *filtering*:



(Texture.cs; line 92 - 97)

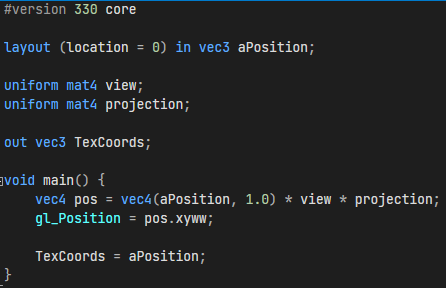
TextureParameterName.TextureWrap{S, T, R} digunakan untuk memberi tahu program bahwa ketiga dimensi dari texture akan diberi metode *wrapping* TextureWrapMode.ClampToEdge.

TextureWrapMode.ClampToEdge digunakan supaya texture di pinggir cubemap tidak muncul garis yang tidak diinginkan.

Setelah semua texture disiapkan, texture Cubemap akan dipasang kepada sebuah cuboid, dengan proses yang sama seperti pemasangan texture yang biasa.

Untuk mengakomodasi kondisi spesial Cubemap, di buatlah *vertex shader* dan *fragment shader* yang baru (cubemap.vert & cubemap.frag).

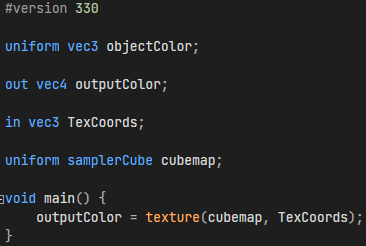
Berikut adalah isi dari cubemap.vert:



Untuk menaikkan performance dari program, cubemap akan digambar terakhir, sehingga saat *depth-buffer* sudah diisi oleh objek lain, hanya bagian - bagian screen yang kosong yang akan digambar. Hal ini akan dicapai oleh penggunaan metode “*Swizzling*”.

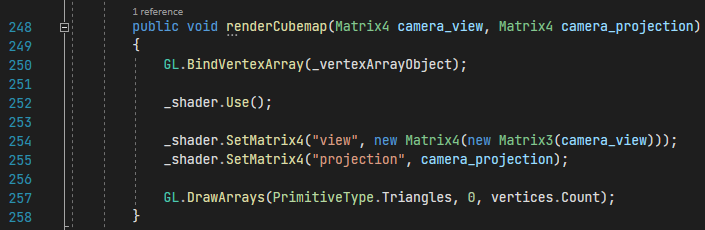
*Swizzling* akan mengganti koordinat dari sebuah vektor dengan koordinat yang ditulis pada *syntax* *swizzle*. Pada potongan code diatas, gl\_Position akan diisi oleh vektor pos, dimana komponen x, y, dan w tidak berubah, tetapi komponen z akan diganti oleh komponen w. Komponen z dari vektor gl\_Position akan digunakan untuk memberi nilai *depth*. Nilai z = 1.0 akan memastikan bahwa cubemap memiliki posisi paling jauh dari posisi kamera, sehingga cubemap akan selalu digambar terakhir.

Berikut adalah isi dari cubemap.frag:



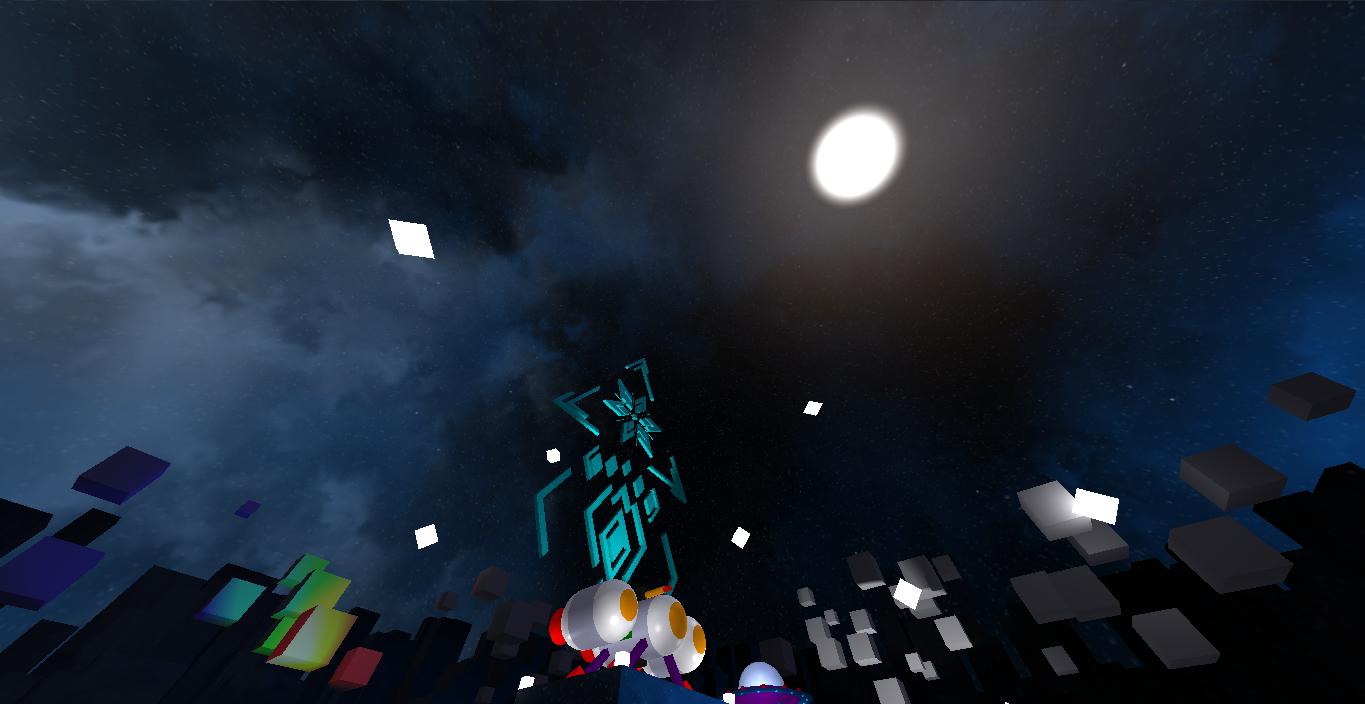
Data texture cubemap akan masuk ke dalam *fragment shader* secara otomatis melalui uniform samplerCube.

Untuk memastikan bahwa hasil akhir skybox tidak bergerak, tetapi selalu mengikuti putaran kamera, view matriks (4 x 4) akan di-*cast* menjadi matriks 3 x 3, kemudian di-*cast* kembali ke matriks 4 x 4. Operasi ini akan menghilangkan komponen translasi dari matriks tersebut, tetapi tidak menyentuh komponen rotasi.



(Asset3d.cs; line 248 - 258)

Berikut adalah hasil proyek dengan skybox:



# Instancing

Saat ingin menampilkan objek - objek yang sangat banyak, tetapi memiliki data *vertex* yang sama, jumlah *draw call* yang dilakukan oleh CPU akan memakan banyak waktu, karena seluruh perintah yang dikirim oleh CPU ke GPU (Pemberitahuan buffer data yang akan dibaca, lokasi *vertex attributes*, dst.) akan melalui bus yang relatif pelan. Walaupun GPU dapat menampilkan objek - objek tersebut dengan cepat, memberikan perintah kepada GPU untuk menampilkan objek yang sangat banyak akan memakan waktu (Vries, n.d.).

Instancing adalah teknik dimana banyak objek (dengan data *vertex* yang sama) akan digambar secara bersamaan. Teknik ini akan menyimpan banyak waktu karena jumlah komunikasi antar CPU dan GPU menurun drastis (Vries, n.d.).

Untuk menggunakan Instancing, *render call* yang digunakan adalah GL.DrawArraysInstanced() atau GL.DrawElementsInstanced(). Function - function tersebut menerima tambahan parameter instanceCount yang memberitahu GPU jumlah objek yang ingin ditampilkan (Vries, n.d.).

Karena terdapat batasan pada jumlah *uniform* yang dapat digunakan secara bersamaan, data matriks transformasi untuk setiap instance dari objek akan dimasukkan ke dalam *vertex attribute* (Vries, n.d.).

Terdapat sedikit perbedaan saat memasukkan mat4 kedalam *vertex attribute*. Jumlah data maksimal yang dapat dimasukkan ke dalam *vertex attribute* adalah sama dengan vec4. Sehingga, mat4 akan menggunakan empat *vertex attribute* untuk menyimpan data matriks (Vries, n.d.).

Berikut adalah code yang memasukkan data mat4 untuk *model matrix* ke dalam *vertex attribute* pada class Asset3dInstanced.cs:



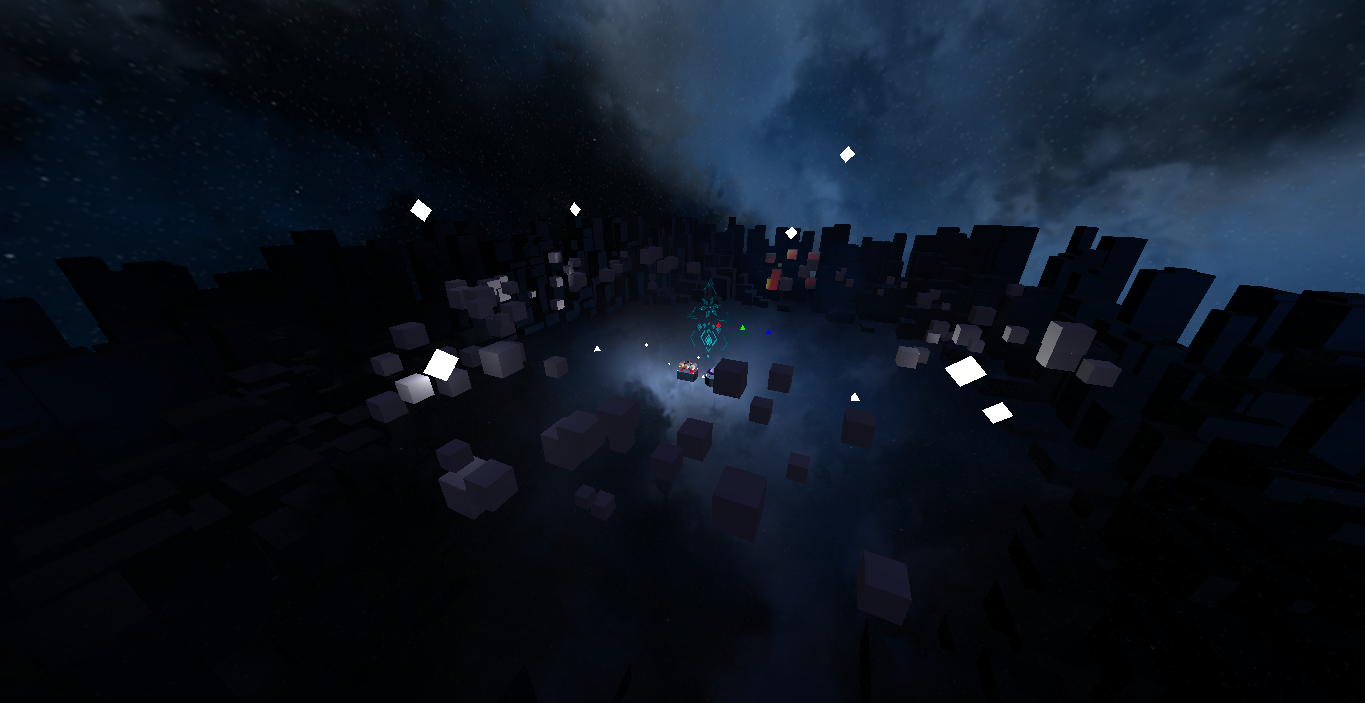
(Asset3dInstanced.cs; line 143 - 169)

GL.VertexAttribDivisor memiliki dua parameter, yaitu:

* Parameter pertama: *vertex attribute* yang bersangkutan
* Parameter kedua: digunakan untuk memberitahu OpenGL bahwa isi dari *vertex attribute* akan berubah setiap kali instance baru akan ditampilkan.

Terdapat juga operasi yang sama pada *normal matrix* untuk seluruh *instanced object* (*vertex attribute* 6 - 9).

Dengan teknik *Instancing*, kelompok kami memutuskan untuk menggambar 100 asteroid berbentuk *Cuboid* dan 500 gunung abstrak berbentuk *cuboid* yang mengelilingi lingkungan proyek.



# Blinn-Phong Shading

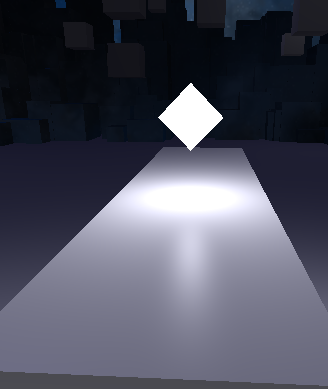
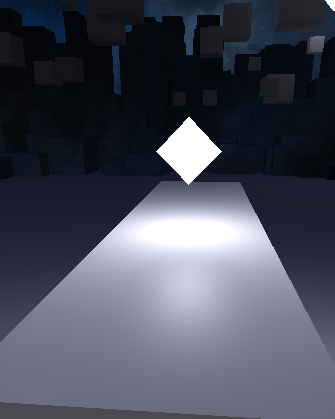
Kelompok kami memutuskan untuk menggunakan Blinn-Phong shading model dan Phong shading model secara bersamaan, dan dapat di-*toggle* dengan menggunakan tombol B pada keyboard. Blinn-Phong memiliki kelebihan yaitu *specular highlights* dari pantulan cahaya terlihat lebih realistis, dimana bentuk dari *specular highlights* akan menjadi lonjong semakin landai sudut pandang kamera terhadap posisi *specular highlights*.

Perbedaan antara Blinn-Phong dan Phong adalah pada perhitungan vektor yang akan di *dot product* untuk menghasilkan *specular highlights*.

Phong shading menggunakan *dot product* dari vektor pandang kamera dan vektor refleksi, dimana vektor refleksi didapatkan dari memantulkan vektor arah cahaya terhadap vektor normal dari sebuah permukaan.

Sedangkan, Blinn-Phong shading menggunakan *dot product* dari vektor normal dan *halfway vector*, dimana *halfway vector* adalah vektor yang memiliki arah persis diantara vektor arah cahaya dan vektor pandang kamera.

Eksponen *shininess* untuk Blinn-Phong shading di-set 2-4x lebih tinggi untuk mendapatkan *specular highlights* yang mirip dengan Phong shading. Untuk proyek kami, Phong shading memiliki eksponen 32, dan Blinn-Phong shading memiliki eksponen 96.



Perbandingan antara Phong (kiri) dan Blinn-Phong (kanan). Kelonjongan *specular highlights* dari Blinn-Phong shading terlihat dengan jelas.

# Kesulitan yang Dialami

Berikut adalah kesulitan yang kelompok kami alami dalam membuat proyek ini:

* Mengimplementasi code OpenGL yang ditulis dalam C++ menjadi C#

Mayoritas tutorial tentang implementasi OpenGL menggunakan bahasa pemrograman C++, sehingga kelompok kami mengalami kesulitan saat mengimplementasi fitur - fitur OpenGL yang lain ke dalam proyek.

* + Solusi

Kelompok kami berhasil menemukan dokumentasi OpenTK dan beberapa jawaban di Stack Overflow tentang fitur - fitur yang kami ingin implementasikan (Cubemap).

* Menghitung *normal vector* dari objek - objek yang dibuat

Pembuatan *normal vector* tidak diajarkan di kelas, sehingga kelompok kami harus mencari *resources* di internet tentang *normal vector* dari sebuah objek.

* + Solusi

Dengan mereview kembali materi mata kuliah Matematika Dasar / Kalkulus I tentang Turunan, kelompok kami berhasil menambahkan *normal vector* kepada objek - objek yang kami buat.

* Menghitung BPM yang tepat untuk animasi dan gerakan kamera

Saat animasi berjalan, pergantian mode animasi menghasilkan *delay* yang dapat dilihat saat animasi berjalan seiringan dengan musik yang kami pilih. Hal ini mengakibatkan animasi untuk berjalan lebih lambat daripada musik.

* + Solusi

Kelompok kami memutuskan untuk menaikkan BPM yang digunakan dalam program dari 145,0 menjadi 145,5. Hal ini mengurangi *delay* yang terjadi secara drastis.

# Keunggulan Proyek

Proyek kami memiliki keunggulan sebagai berikut:

* Tidak menggunakan file .obj

Semua objek yang ada di dalam proyek dibuat secara matematis, termasuk *normal vector* dari seluruh objek.

* Penggunaan Cubemap

Dengan menggunakan Cubemap untuk mengimplementasikan sebuah skybox, lingkungan dari proyek terkesan luas dan lebih imersif daripada background monotone.

* Penggunaan Instancing

Instancing digunakan untuk meletakkan ratusan atau lebih objek yang sama dengan transformasi masing - masing. Dengan Instancing, pemanggilan fungsi render tidak mencapai jumlah yang dapat mempengaruhi performance dari proyek secara negatif.

* Penggunaan Blinn-Phong Shading Model

Blinn-Phong Shading digunakan untuk memberikan pantulan cahaya yang lebih realistis. Komponen *specular* dari cahaya menggunakan Blinn-Phong shading lebih mendekati pantulan cahaya yang terjadi di dunia nyata.

# Penutup

Demikian laporan yang kami buat, kami berharap bahwa laporan ini telah menjelaskan proyek kami dengan jelas. Kami mohon maaf apabila ada kesalahan kata dan ejaan yang kurang tepat. Atas waktunya kami ucapkan terima kasih sebesar-besarnya.

# Daftar Pustaka

Köller, J. (2001). *Tetrahedron*. Retrieved from http://www.mathematische-basteleien.de/tetrahedron.htm

Vries, J. (n.d.). *Camera*. Retrieved December 16, 2021 from https://learnopengl.com/Getting-started/Camera

Vries, J. (n.d.). *Light casters*. Retrieved December 16, 2021 from https://learnopengl.com/Lighting/Light-casters

Vries, J. (n.d.). *Cubemaps*. Retrieved December 15, 2021 from https://learnopengl.com/Advanced-OpenGL/Cubemaps

Vries, J. (n.d.). *Instancing*. Retrieved December 15, 2021 from https://learnopengl.com/Advanced-OpenGL/Instancing

Lawrence, J. (1972). *A catalog of special plane curves.* New York: Dover Publications.