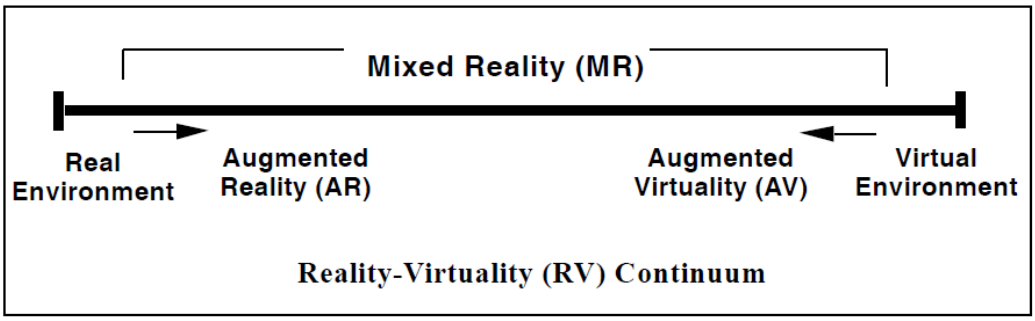
BAB II
LANDASAN TEORI

2.1 Augmented Reality

Augmented Reality (AR) adalah penggabungan benda-benda nyata dan maya dilingkungan nyata, berjalan secara interaktif dalam waktu nyata, dan terdapat integrasi antarbenda dalam tiga dimensi, yaitu benda maya terintegrasi dalam dunia nyata [1]. Penggabungan benda nyata dan maya dimungkinkan dengan teknologi tampilan yang sesuai, interaktivitas dimungkinkan melalui perangkat-perangkat input tertentu [2]. Paul Milgram dan Fumio Kishino mendifinisikan Milgram's Reality-Virtual Continuum pada tahun 1994. Mereka mendeskripsikan sebuah kontinum yang membentang dari lingkungan nyata ke lingkungan virtual murni. Diantaranya terdapat *augmented reality* (lebih dekat ke lingkungan nyata) dan *augmented virtuality* (lebih dekat ke lingkungan virtual) [5].



Gambar 2. 1 Simplified representation of a RV Continuum

Gambar 2.1 menggambarkan *augmented reality* (AR) lebih dekat ke sisi kiri yang berarti lingkungan bersifat nyata dan benda bersifat maya, sedangkan *augmented virtuality* lebih dekat ke sisi kanan yang berarti lingkungan bersifat maya dan benda bersifat nyata. Keduanya digabungkan menjadi *mixed reality* (MR).

Augmented reality memiliki beberapa komponen yang mendukung dalam proses pengolahan citra digital [6], diantaranya adalah :

a. Scene Generator

Scene generator adalah komponen yang bertugas untuk melakukan *rendering* citra yang ditangkap oleh kamera. Objek virtual akan ditangkap kemudian diolah sehingga dapat ditampilkan.

b. Tracking System

Tracking system merupakan komponen yang terpenting dalam *augmented reality*. proses *tracking* dilakukan untuk mendeteksi pola objek virtual dan objek nyata sehingga terjadi sinkronisasi diantara keduanya dalam hal ini proyeksi virtual dengan proyeksi nyata harus sama atau mendekati sama sehingga mempengaruhi validitas hasil yang akan didapatkan.

c. Display

Pembangunan sebuah sistem yang berbasis AR dimana sistem tersebut menggabungkan antara dunia virtual dan dunia nyata ada beberapa parameter mendasar yang perlu diperhatikan yaitu optik dan teknologi video. Keduanya mempunyai keterkaitan yang tergantung pada faktor resolusi, fleksibiltas, titik pandang, *tracking* area. Terdapat batasan-batasan dalam pengembangan teknologi *augmented reality* dalam hal proses menampilkan objek. Diantaranya adalah harus ada batasan pencahayaan, resolusi layar, dan perbedaan pencahayaan citra antara citra virtual dan nyata.

2.1.1 Teknik Tracking Pada Augmented Reality

Seiring dengan perkembangannya, *augmented reality* memiliki beberapa teknik dalam proses *tracking*, diantaranya adalah sebagai berikut :

a. Marker Based Tracking

Teknik ini menggunakan *marker* hitam dan putih persegi dengan batas hitam tebal dan latar belakang berwarna putih. Komputer akan mengenali posisi dan orientasi objek *marker* tersebut dan menciptakan sebuah dunia virtual 3D yaitu titik (0,0,0) dan sumbu yang terdiri dari x, y dan z. *Marker based tracking* ini sudah

lama dikembangkan sejak tahun 1980 an. *Library* atau SDK yang dapat digunakan untuk teknik ini adalah ARToolkit.

b. Markerless Tracking

Teknik ini sebenarnya masih menggunakan *marker* sebagai alat untuk mendeteksi objek, namun *marker* yang digunakan berbeda dengan teknik *marker based tracking* yang menggunakan *marker* persegi berwarna hitam dan putih. *Marker* yang digunakan untuk proses *tracking* metode ini menggunakan fitur alami (*natural features*) dari gambar atau objek yang akan dilacak, seperti tepi, sudut, atau sehingga metode ini dapat menghasilkan proses *tracking* yang lebih kuat dan cocok untuk digunakan pada *mobile augmented reality* [3]. *Library* atau SDK yang dapat digunakan untuk teknik ini adalah Layar, IN2AR, D'Fusion Studio, Vuforia Qualcomm.

c. Face Tracking

Teknik ini menggunakan algoritma untuk dapat mengenali wajah manusia secara umum dengan cara mengenali posisi mata, hidung dan mulut yang kemudian akan mengabaikan objek-objek lain disekitarnya.

d. Motion Tracking

Teknik ini melakukan deteksi dengan cara menangkap gerakan dan telah mulai digunakan untuk memproduksi film-film yang mensimulasikan gerakan pada tubuh.

e.* GPS *Tracking

Teknik GPS *tracking* memanfaatkan lokasi *longitude* dan *latitude* sebagai alat untuk mendeteksi objek secara *realtime*. Teknik ini biasanya digunakan pada perangkat smartphone yang memiliki fitur GPS.

f. 3D *Object Tracking*

Teknik ini menggunakan benda nyata 3D sebagai alat pemindai seperti mobil, motor, meja dan lain-lain.

2.2 Vuforia Qualcomm

Vuforia Qualcomm merupakan *library* yang digunakan sebagai pendukung adanya *augmented reality* pada sistem operasi android dan IOS [10]. Vuforia

menganalisa gambar dengan menggunakan pendeteksi *marker* dan menghasilkan informasi 3D dari *marker* yang sudah dideteksi melalui API. *Programmer* juga dapat menggunakannya untuk membangun objek 3D virtual pada kamera.

Beberapa keuntungan yang didapatkan ketika mengembangkan aplikasi menggunakan vuforia, diantaranya :

a. Deteksi target lokal lebih cepat
b. Pelacakan yang kuat dan tidak mudah hilang ketika perangkat digerakan
c. Hasil yang lebih baik dalam kondisi cahaya yang rendah
d. Optimasi yang menjamin grafis yang lebih baik dan lebih realistis pada saat rendering
e. Pelacakan simultan hingga lima target

Adapun contoh nyata pembuatan objek 3D dengan menggunakan vuforia adalah seperti ditunjukkan pada gambar 2.2 [7]



Gambar 2. 2 Objek Mobil 3D Virtual

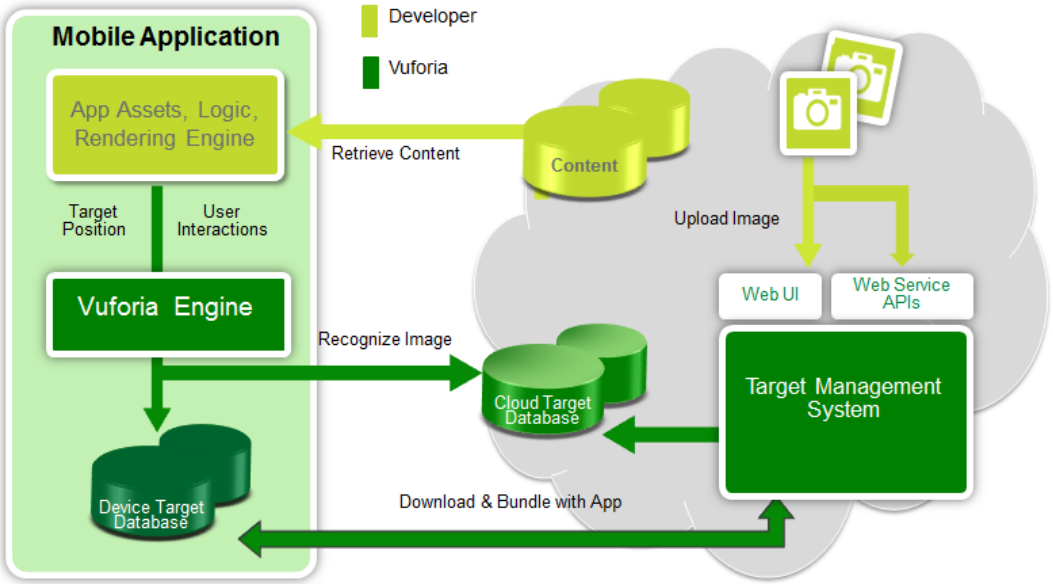
Gambar 2.2 adalah pengembangan aplikasi menggunakan palform Qualcomm AR. Platform tersebut terdiri dari 2 komponen [7] diantaranya adalah :

1. *Target Management System*

Mengijinkan para pengembang melakukan upload gambar yang sudah diregistrasi oleh *marker* dan kemudian melakukan *download target* gambar yang akan dimunculkan.

2. QCAR SDK Vuforia

Mengijinkan para pengembang untuk melakukan koneksi antara aplikasi yang sudah dibuat dengan *library static*, contohnya libQCAR.a pada IOS atau libQCAR.so pada android [10].



Gambar 2. 3 QCAR SDK library

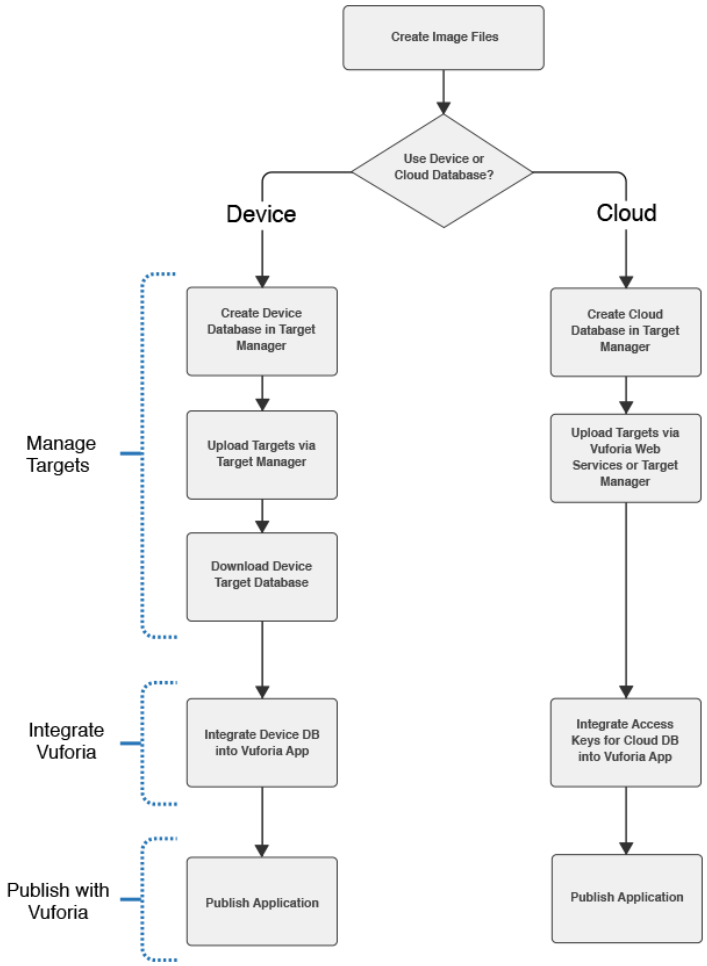
Gambar 2.3 memberikan gambaran umum pembangunan aplikasi dengan Qualcomm AR Platform. Platform ini terdiri dari SDK QCAR dan *Target System Management* yang dikembangkan pada portal QdevNet. *User* meng-upload gambar masukan untuk *target* yang ingin dijadikan *marker* dan kemudian men-download sumber daya target, yang dipackage dengan *app*. SDK QCAR menyediakan sebuah objek yang terbagi libQCAR.so yang harus dikaitkan dengan *app*.

Pembuatan target terbagi menjadi 2 pilihan, yaitu menggunakan *device based databases* atau *cloud based databases*. Perbedaannya dapat dilihat pada tabel 2.1

Tabel 2. 1 Target Database QCAR

Device Database	*Cloud Database*
Maksimal 100 target dalam suatu target *database*	Lebih dari 1 juta target dalam *cloud database*
Dapat mendownload target yang berbeda-beda	Satu *database* untuk seluruh gambar dan *metadata*
Tidak *support metadata*	*Support metadata*
Tidak membutuhkan koneksi internet	Dibutuhkan koneksi internet
Gratis	Gratis dan berbayar, tergantung penggunaan.

Langkah pembuatan yang lebih jelasnya dapat dilihat pada gambar 2.4 [10].



Gambar 2. 4 Alur Diagram QCAR

Gambar 2.4 menggambarkan secara umum bagaimana alur dalam pembuatan aplikasi *augmented reality* menggunakan *library* vuforia. Berawal dengan pembuatan *file* gambar untuk keperluan *marker* kemudian memutuskan apakah menggunakan *device database* atau *cloud database*. Sampai pada akhirnya melakukan publish aplikasi.

2.2.1 Image Target

Image target merupakan gambar yang dapat dideteksi dan dilacak oleh vuforia SDK. Tidak seperti *marker* tradisional, kode matriks data, dan kode QR, *image target* tidak perlu berwarna hitam dan putih atau memiliki kode untuk dapat dikenali [10]. Vuforia SDK menggunakan algoritma yang canggih untuk mendeteksi dan melacak fitur secara alami yang ditemukan didalam gambar itu sendiri. vuforia sdk mengenali *image target* dengan cara membandingkan fitur alami pada image target terhadap *database resource* yang telah didaftarkan (diupload) sebelumnya. Setelah *image target* terdeteksi, SDK akan melacak gambar menggunakan kamera *device*.

Image target dapat dibuat dan digunakan dalam tiga cara yang berbeda, yaitu:

1. Membuat *image target* dengan *Target Manager* yang digunakan dalam perangkat *database*
2. Membuat *image target* menggunakan *cloud recognition databases*
3. *Image target* dibuat pada saat aplikasi sedang berjalan melalui kamera dan secara dinamis ditambahkan pada perangkat *database*.

Image Target dapat dibuat secara *online* melalui *Target Manager* dari gambar bertipe JPG atau PNG (hanya gambar RGB atau *grayscale* yang *support*) dengan ukuran maksimal 2MB. fitur diekstrak dari gambar-gambar yang disimpan dalam *database* dan kemudian dapat didownload dan digunakan dalam aplikasi. *Database* kemudian dapat digunakan oleh vuforia untuk perbandingan pada saat aplikasi dijalankan. Pada saat *runtime*, vuforia SDK dapat mendeteksi dan melacak hingga lima target secara simultan. Performanya akan bergantung pada beban *processor* dan GPU [10].

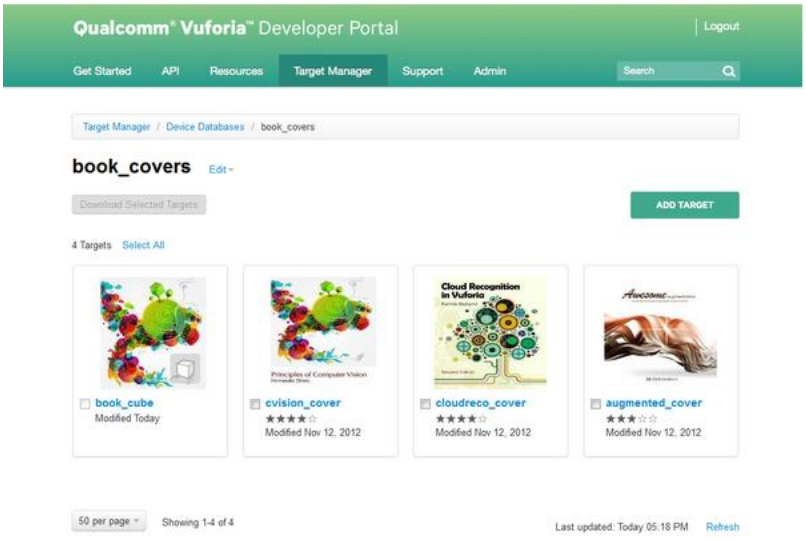
A. Membuat *Image Target*

Membuat *image target*, kualitas merupakan hal yang penting agar aplikasi dapat hasil yang optimal. Attribut yang ideal untuk sebuah *image target* meliputi :

a. Kaya akan *detail*, contohnya gambar pemandangan jalan, sekelompok orang dan lain-lain
b. Kontras yang baik, yaitu daerah terang dan gelap
c. Tidak ada pola yang berulang, contohnya lapangan rumput
d. harus 8- atau 24-bit PNG dan JPG format, kurang dari 2 MB, JPG harus RGB atau *grayscale*

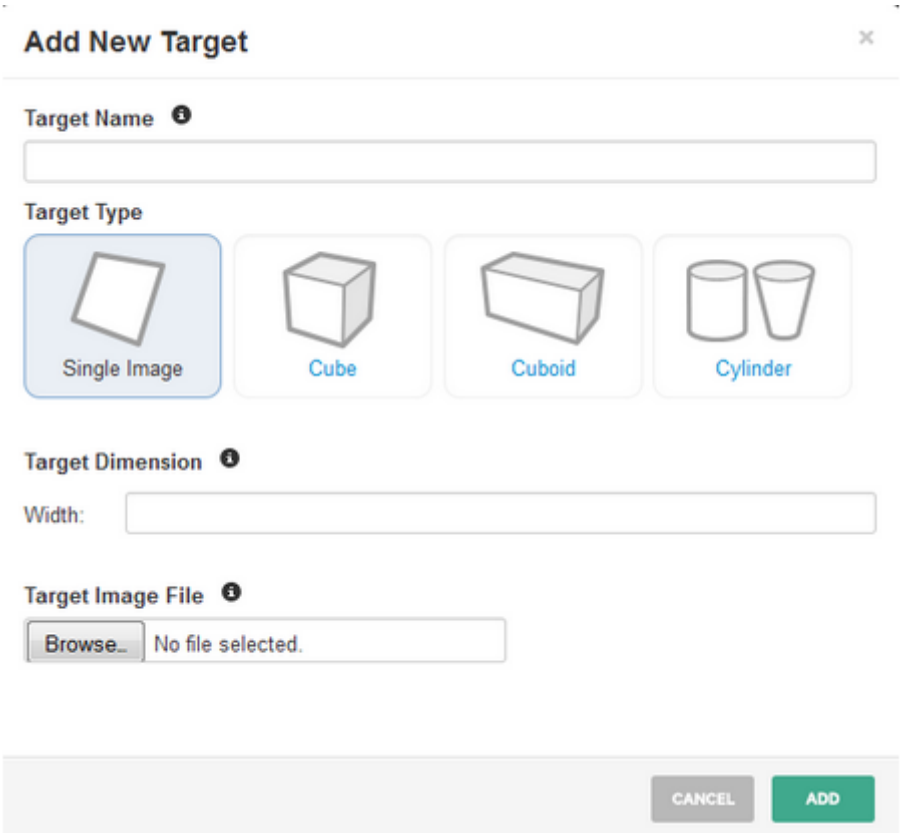
B. Menambahkan *Image Target* pada *Database*

1. Buka halaman utama *Target Manager* melalui *browser* https://developer.vuforia.com/target-manager
2. Kemudian *log in*, jika belum memiliki *account* dapat melakukan registrasi terlebih dahulu.
3. Kemudian klik *tab device databases*. *web* akan menampilkan *database* yang telah dibuat sebelumnya (jika sudah pernah membuat).
4. Buatlah sebuah *database* baru dengan mengklik tombol *create database*, lalu masukan nama *database*.
5. Pilihlah *database* yang akan digunakan seperti yang terlihat pada gambar 2.5.



Gambar 2. 5 Contoh Target Manager Website Vuforia

6. Kemudian pilih tombol *add target* untuk menambahkan *image target*.



Gambar 2. 6 Kotak Dialog Target Manager

Gambar 2.6 merupakan tampilan kotak dialog target manager yang berisi informasi mengenai *target image* yang akan dibuat.

7. Ketikan nama untuk target pada kolom *target name* dengan kriteria sebagai berikut : nama harus unik dalam satu *target database*, nama hanya dapat berisi huruf, angka, dan karakter garis bawah '_'.
8. Pilih *target type* untuk *image target* sesuai dengan kebutuhan.
9. Masukkan ukuran lebar dari target yang akan dibuat.
10. Klik tombol *Browse* dan pilih gambar yang akan dijadikan target. *File* diharuskan 8 atau 24-bit PNG atau JPG. *File* JPG diharuskan RGB atau *grayscale*. Ukuran *file* gambar maksimum 2,25MB.
11. Klik tombol *Add*. Setelah diproses, Halaman *target detail* akan ditampilkan dengan hasil fitur beserta ratingnya. Image Target memiliki beberapa parameter, diantaranya dijelaskan pada tabel 2.2.

Tabel 2. 2 Image Target Parameter

Target Size	a. Ukuran sebenarnya dari *image target* dalam 3D *scene unit* b. Pengembang harus menentukan ukuran target saat membuat target *online* atau di konfigurasi *Database file* XML c. *Database Configuration* XML *file* dihasilkan oleh *Target Manager*, tetapi setelah dibuat dapat dimodifikasi oleh pengembang. d. Parameter ukuran target sangat penting, karena mengembalikan informasi selama pelacakan berada diskala yang sama, misalnya jika lebar gambar 16 unit, menggerakan kamera dari batas kiri target ke batas kanan target perubahan posisi dikembalikan 16 unit sepanjang sumbu x. (x,y) – ukuran target unit diukur sepanjang sumbu horisontal dan vertikal dari target persegi panjang.
Virtual Buttons	a. *Image Target* dapat memiliki satu atau lebih *virtual button* b. *Virtual button* dapat secara dinamis ditambahkan atau dihapus saat aplikasi dijalankan.
Unique Target ID	a. Sebuah sistem identifikasi unik yang besar untuk mengindentifikasi target gambar diseluruh *cloud* dan setiap *dataset* lainnya.

C. Mencetak Target

a. Ukuran

Ukuran meja, area yang dekat, rak produk dan sejenisnya, target gambar dicetak setidaknya 5 *inci* atau 12cm untuk lebar dan tinggi agar mendapatkan AR *experience* yang baik. Ukuran yang direkomendasikan bervariasi berdasarkan *rating* target dan jarak ke *image target*.

b. Kerataan Bidang

Kualitas pelacakan menggunakan vuforia SDK dapat menurun secara signifikan ketika target yang dicetak tidak datar. Ketika membuat cetakan target, pastikan bahwa target tidak menekuk dan tidak berkerut. Sebuah trik sederhana dengan menggunakan kertas tebal saat mencetak yaitu dengan kertas 200-220 gram.

c. Penyelesaian

Cetakan dari *printer* laser modern mungkin akan membuat target menjadi sangat mengkilap. Dalam kondisi pencahayaan yang normal mungkin tidak akan masalah, namun bila terdapat sumber cahaya (lampu, sinar matahari,

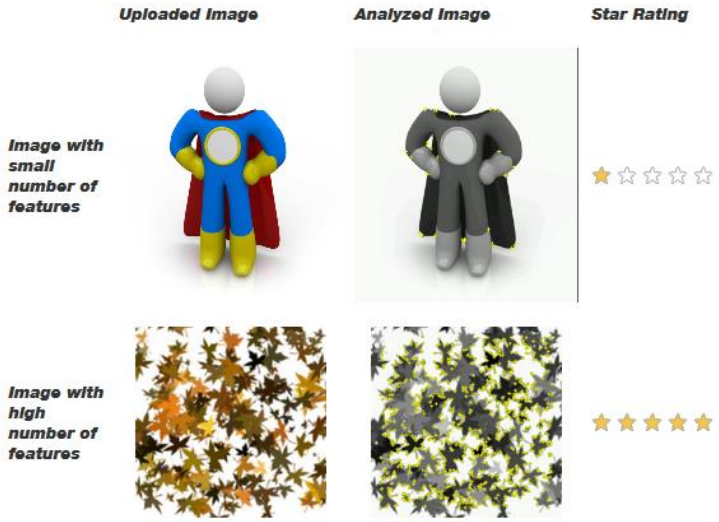
dll) dari sudut tertentu akan membuat pantulan cahaya yang berpotensi menutupi sebagian besar tekstur asli dari hasil cetakan dan hal ini dapat membuat masalah saat proses pelacakan.

2.2.2 Natural Features and Rating

Rating dalam hal ini mendefinisikan seberapa baik gambar dapat dideteksi dan dilacak menggunakan SDK Vuforia. Penilaian ini ditampilkan pada *Target Manager* setelah gambar yang akan dijadikan target diupload melalui API di web vuforia. pemberian *rating* berkisar dari 0 sampai 5 untuk setiap gambar. Semakin tinggi *rating* dari dari gambar target, maka semakin kuat kemampuan deteksi dan pelacakannya yang dikandungnya. rating nol menunjukan bahwa target tidak dapat dilacak sama sekali oleh sistem AR, sedangkan *rating* bintang 5 menunjukan bahwa gambar mudah dilacak oleh sistem AR.

A. *Features*

Sebuah fitur merupakan tajam, berduri, detil dipahat dalam gambar, seperti yang ada pada benda-benda bertekstur. *Image analyzer* mengumpamakan fitur sebagai tanda panah kecil berwarna kuning. Tingkatkan detail pada gambar yang ingin digunakan dan pastikan bahwa rincian tidak membuat pola yang berulang. Melihat lebih jelas perbandingan antara gambar yang memiliki banyak fitur dengan gambar yang memiliki sedikit fitur dapat dilihat pada gambar 2.7

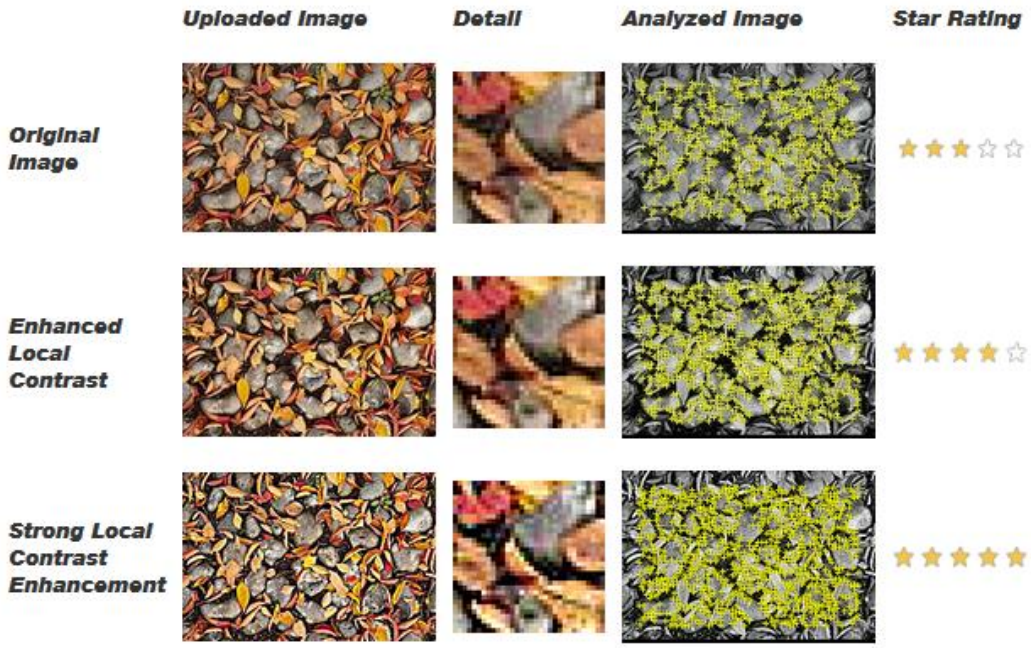


Gambar 2. 7 Perbandingan Features

Gambar 2.7 memperlihatkan fitur yang dihasilkan dari gambar yang memiliki sudut tajam dan detail yang banyak akan menghasilkan fitur yang banyak juga dan hal itu pun akan berpengaruh pada rating yang didapatnya.

B. *Local Contrast*

Baik atau buruknya *local contrast* seringkali sulit untuk dideteksi dengan mata saja. Meningkatkan kontras gambar secara umum, atau memilih gambar dengan sisi yang lebih rinci. Bentuk organik, detail bulat, *blur*, atau gambar yang dikompresi sering tidak memberikan hasil yang cukup secara rinci untuk dideteksi dan dilacak dengan benar.



Gambar 2. 8 Pengaturan Kontras

Gambar 2.8 menjelaskan bahwaa dengan melakukan pengaturan pada kontras gambar dapat meningkatkan jumlah fitur yang dihasilkan sehingga meningkatkan rating gambar tersebut.

C. *Feature Distribution*

Semakin seimbang distribusi fitur dalam gambar, semakin baik gambar tersebut dapat dideteksi dan dilacak. Pastikan bahwa fitur terdistribusi di seluruh

gambar. Pertimbangkan untuk melakukan *cropping* pada sebuah gambar untuk menghapus area tanpa fitur seperti pada gambar 2.9

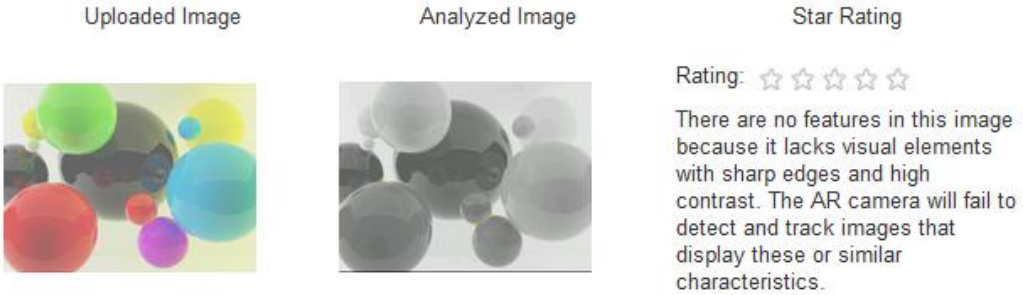


Gambar 2. 9 Distribusi Fitur

Gambar 2.9 Menggambarkan bahwa salah satu cara yang dapat dilakukan untuk mendapatkan distribusi fitur yang seimbang dalam sebuah gambar adalah dengan cara *cropping* pada bagian yang menghasilkan banyak fitur dan itu akan jauh lebih baik dalam pelacakannya.

D. *Avoid Organic Shapes*

Biasanya, bentuk-bentuk organik yang halus atau rincian yang mengandung aspek kabur/*blur* atau kompresi yang sangat tinggi tidak memberikan detail yang cukup untuk dideteksi dan dilacak dengan benar dan memiliki jumlah fitur yang rendah.

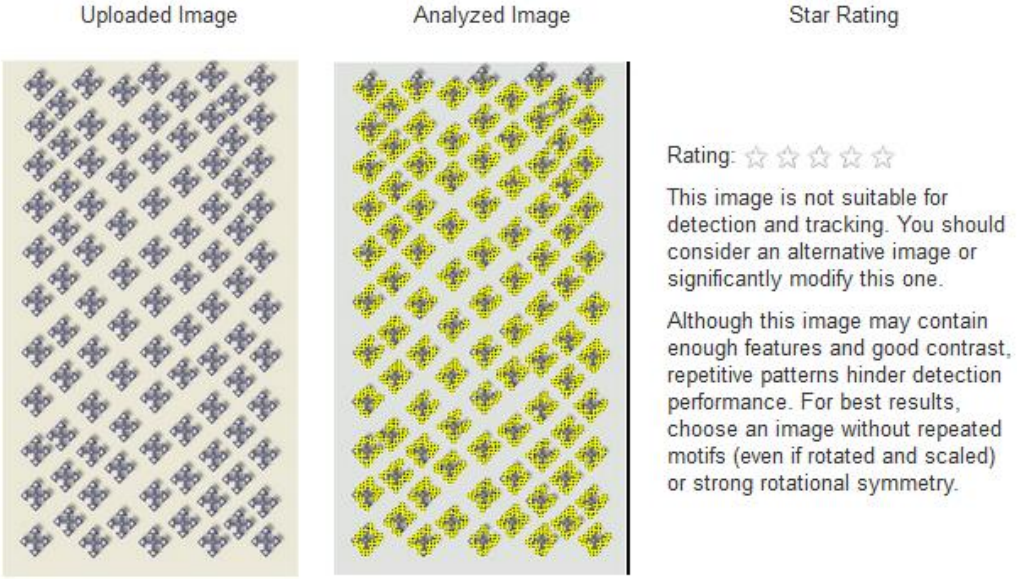


Gambar 2. 10 Bentuk Organik

Gambar 2.10 menggambarkan target dengan bentuk organik tidak memiliki fitur yang cukup untuk dideteksi dan dilacak.

E. *Avoid Repetitive Patterns*

Meskipun beberapa gambar mengandung cukup fitur dan kontras yang baik, pola berulang akan menghambat kinerja deteksi. Untuk hasil terbaik, pilih gambar tanpa motif berulang (bahkan jika diputar dan diperbesar) atau simetri rotasi yang kuat. Gambar kotak-kotak adalah contoh dari pola berulang yang tidak dapat dideteksi, karena pasangan 2x2 kotak hitam dan putih tampak persis sama dan tidak dapat dibedakan oleh detektor.

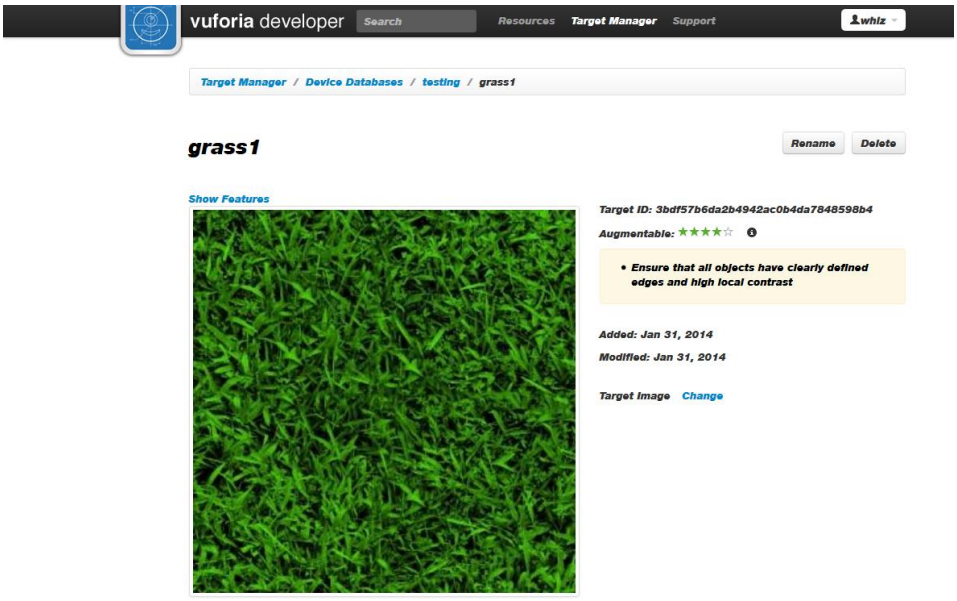


Gambar 2. 11 Pola Berulang

Gambar 2.11 menggambarkan bahwa gambar yang menghasilkan banyak fitur tapi memiliki pola yang berulang tidak dapat dideteksi oleh sistem (tidak memiliki *rating*).

2.2.3 Target Management System

Qualcomm *target management system* mengijinkan pengembang untuk melakukan upload gambar dan menghasilkan kumpulan data pada target tujuan [7]. Aplikasi dapat mencocokkan *image* dalam *frame* berikut dengan kumpulan datanya.



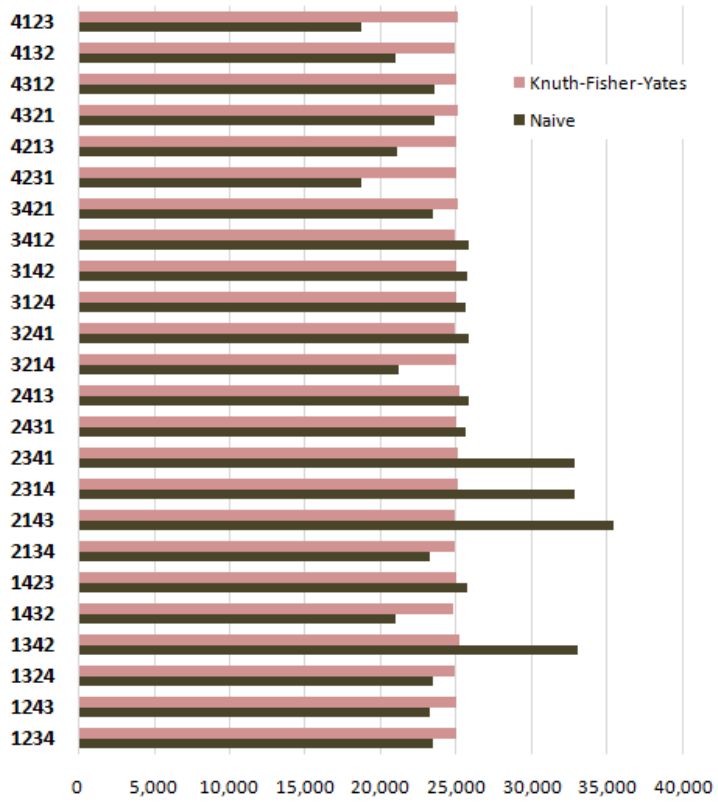
Gambar 2. 12 Target Management System

Gambar 2.12 merupakan tampilan website vuforia yang disediakan oleh qualcomm untuk keperluan proses registrasi marker.

2.3 Algoritma Fisher-Yates Shuffle

Fisher-yates *shuffle* merupakan algoritma yang digunakan untuk mengubah urutan masukan yang diberikan secara acak. Permutasi yang dihasilkan oleh algoritma ini muncul dengan probabilitas yang sama.

Algoritma ini dinyatakan bias karena permutasi yang dihasilkan oleh algoritma ini muncul dengan probabilitas yang sama, hal ini dibuktikan dengan percobaan mengacak suatu set kartu yang dilakukan berulang-ulang seperti yang bisa dilihat pada gambar 2.13 [11].



Gambar 2. 13 Hasil Perbandingan Algoritma Shuffle

Pada gambar 2.13 dilakukan tes terhadap 4 kartu, yang kemudian diacak sebanyak 600.000 kali. Keterangan pada sumbu y menunjukan permutasi/ kemungkinan kombinasi kartu yang muncul, sedangkan keterangan pada sumbu x menunjukan jumlah kemunculan kombinasi yang muncul. Warna pink menunjukan hasil dari algoritma fisher-yates *shuffle* sedangkan warna hitam menunjukan algoritma naïve *shuffle*. Hasilnya algoritma fisher-yates menghasilkan nilai yang hampir sama untuk setiap kemungkinan kombinasi kartu, sedangkan pada algoritma naïve kombinasi kartu tertentu muncul jauh lebih sering dan sebagian lagi muncul lebih sedikit dibanding kombinasi kartu lain.

Metode fisher-yates shuffle secara umum adalah:

1. Ketika masih ada elemen tersisa untuk diacak
2. Ambil elemen secara acak dari elemen yang tersisa
3. Kemudian tukar dengan elemen saat ini

Pseudocode dari metode fisher-yates shuffle adalah sebagai berikut:

```
function fisherYates(array)
    for i in <array.length - 1, 1> do
    index = random number from interval <0, i)
    swap(array[i], array[index])
  end
```

2.4 Collision Detection

Collision detection atau deteksi tumbukan merupakan suatu hal mendasar dalam bidang *game programming* dan hampir sebagian besar *game* membutuhkan hal ini sebagai inti dari *game mechanic*. *Collision detection* adalah fungsi yang digunakan untuk dapat mengetahui obyek-obyek apa saya yang bersentuhan satu sama lain dalam bidang koordinat 2 dimensi ataupun 3 dimensi. Selain itu, *collision detection* juga berguna untuk menentukan posisi dari satu obyek dengan obyek yang lain sehingga tidak ada obyek yang saling menembus [9].

Banyak game yang menggunakan *collision* detection sebagai penunjang dari mekanika gamenya contohnya, *game* dengan tipe *shooting game* atau tembak-tembakan, *collision detection* pada *game* dengan tipe seperti ini digunakan untuk mengetahui apakah peluru yang ditembakan oleh pemain mengenai lawan atau obyek lain yang ada didalam *game*. Untuk mempercepat proses pada *collision detection*, umumnya obyek-obyek direpresentasikan secara *logic* dengan bentuk primitif seperti segi empat dan lingkaran jika pada koordinat dua dimensi, atau kubus dan bola jika pada koordinat tiga dimensi.

Collision detection memiliki teknik yang berbeda untuk diterapkan pada pembuatan *game* 2D dan 3D.

a. Teknik *collision detection* pada obyek 2D memiliki 2 cara yaitu:

1. *Rectangles Collision Detection*

   Teknik ini merupakan pendeteksian tabrakan menggunakan *bound* berbentuk persegi, dimana setiap *sprite* atau obyek yang dibuat dalam *game* memiliki *bound* masing-masing. Cara ini cukup mudah karena hanya perlu membandingkan apakah ada *intersection* (irisan) kedua obyek pada koordinat

tertentu. Namun cara ini kurang akurat karena untuk *sprite* atau obyek yang memiliki ruang kosong yang besar dengan *bound*. Hal ini akan muncul saat kondisi kedua batas obyek tersebut beririsan walaupun sebenarnya tidak terjadi tabrakan maka akan dianggap terjadi tabrakan. Cara ini cocok untuk *game* yang tidak memperhatikan akurasi deteksi tabrakan atau untuk *game* yang menggunakan sprite yang bentuknya mendekati persegi.

Adapun cara lain dari hasil modifikasi teknik ini yaitu *reduced size bounding box*. Dimana kotak batasnya diperkecil dengan mengurangi beberapa *pixel* dari ukuran *sprite atau* obyek yang sebenarnya. Sehingga dengan cara ini pada saat melakukan *collision detection* akan lebih akurat karena ruang kosong antara *sprite* dan *bound* berkurang.

Kemudian cara lain dari modifikasi teknik ini adalah dengan cara menggunakan beberapa *bounding box* secara bersamaan untuk satu *sprite* atau obyek. Contohnya untuk *sprite* berbentuk manusia, pertama *bounding box* digunakan untuk bagian kepala kemudian *bounding box* yang lainnya digunakan untuk badan dan kaki. Cara ini lebih akurat karena banyaknya *bounding box* yang digunakan dapat disesuaikan dengan bentuk gambar pada sprite. Namun semakin banyak *bounding box* yang digunakan akan semakin banyak pula waktu yang digunakan untuk *collision detection*.

2. *Circle Rectangle Collision Detection*

Untuk teknik ini memiliki cara yang hampir sama dengan yang sebelumnya hanya saja disini *bound* atau batas yang digunakan berbentuk lingkaran. Cara pengecekannya lebih mudah karena linggkaran memiliki jarak yg sama ke batas luarnya. Caranya yaitu dengan menghitung jarak kedua obyek dan membandingkannya dengan jumlah radius masing-masing *bound* dari kedua obyek, jika jaraknya lebih kecil dari jumlah radius kedua obyek maka terjadi tabrakan(*collision*) dan jika lebih besar maka tidak terjadi tabrakan. Cara sebelumnya yaitu *rectangle collision detection* lebih akurat daripada cara ini.

b. Teknik *collision detection* pada obyek 3D memiliki 2 cara yaitu:

1. *Bounding Box Collision Detection*

   Teknik ini menggunakan cara yang hmpir sama dengan teknik *Rectangle Collision Detection* pada 2D hanya batas kotaknya berbentuk 3D uang melingkupi setiap obyek. Untuk melakukan pengecekan apakah kotak-kotak itu saling beririsan atau tidak kita cukup membandingkan nilai maksimum dan minimum pada koordinat x,y dan z.

2. *Sphere Collision Detection*

   Untuk teknik ini bound atau batas yang digunakan berbentuk bola yang melingkupi setiap obyek. *collisions detection* yang dilakukan sebagai berikut :

   Obyek A berpusat di ax,ay,az dan memiliki radius ar

   Obyek B berpusat di bx,by,bz dan memiliki radius br

2.5 Android

Android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware*, dan aplikasi [10]. Android adalah sistem operasi untuk telepon seluler yang berbasis Linux. Android menyediakan *platform* terbuka bagi para pengembang untuk membuat aplikasi mereka sendiri.

Pada awalnya dikembangkan oleh Android Inc, sebuah perusahaan pendatang baru yang membuat perangkat lunak untuk ponsel yang kemudian dibeli oleh Google Inc. Untuk pengembangannya, dibentuklah *Open Handset Alliance* (OHA), konsorsium dari 34 perusahaan perangkat keras, perangkat lunak, dan telekomunikasi termasuk Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, dan Nvidia.

2.5.1 Versi Android

Sistem operasi android memiliki beberapa versi, diantaranya adalah sebagai berikut :

a. Android versi 1.5 (Cupcake)

Android versi 1.5 dirilis pada pertengahan Mei 2009, memiliki beberapa penambahan fitur dalam seluler yakni kemampuan merekam dan menonton *video* dengan modus kamera, mengunggah *video* ke *youtube* dan gambar ke *picasa* langsung melalui telepon.

b. Android versi 1.6 (Donut)

Pada September 2009, Google merilis android versi 1.6 dengan kemampuan menampilkan proses pencarian yang lebih baik dibanding sebelumnya, penggunaan baterai indikator dan kontrol *applet* VPN.

c. Android versi 2.0 (Éclair)

Android versi 2.0 dirilis pada 3 Desember 2009, perubahan yang dilakukan adalah dengan mengoptimalkan *hardware*, peningkatan Google Maps 3.1.2, perubahan UI dengan *browser* baru dan dukungan HTML 5, daftar kontak baru, dukungan *flash* untuk kamera 3,2 MP, *digital zoom*, dan *Bluetooth* 2.1.

d. Android versi 2.2 (Froyo)

Pada 20 Mei 2010, Google merilis android versi 2.2 dengan penambahan dukungan adobe flash 10.1, integrasi V8 *javascript engine* yang dipakai Google Chrome yang mempercepat kemampuan rendering pada browser, pemasangan aplikasi dalam SD Card, kemampuan Wifi Hotspot portable, dan kemampuan auto update dalam aplikasi android market.

e. Android versi 2.3 (Gingerbread)

Android versi 2.3 diluncurkan pada tanggal 6 Desember 2010 dengan peningkatan kemampuan permainan (gaming), peningkatan fungsi copy paste, perubahan UI (User Interface), dukungan kemampuan NFC (Near Field Communication), dan dukungan jumlah kamera lebih dari satu.

f. Android versi 3.0 (Honeycomb)

Android versi ini dirancang khusus untuk perangkat tablet dengan dukungan ukuran layar yang lebih besar. Honeycomb juga mendukung multi proses dan akselerasi perangkat keras untuk grafis.

g. Android versi 4.0 (Ice Cream Sandwich)

Pada tanggal 19 Oktober 2011, android versi 4.0 dirilis dengan membawa fitur dari versi 3.0 dan menambahkan fitur baru diantaranya membuka kunci dengan pengenalan wajah, pengontrol penggunaan jaringan data, sinkronisasi kontak, perangkat tambahan fotografi, dan berbagi informasi dengan menggunakan NFC.

h. Andoid versi 4.1 (Jelly Bean)

Android versi 4.1 dirilis dengan penambahan fitur baru, diantaranya dukungan terhadap OpenGL ES 3.0 yang menjadikan performansi tinggi pada sektor grafis, selain itu terdapat juga fitur bluetooth smart yang dapat menghemat daya pada saat pemakaian bluetooth.

i. Andoid versi 4.4 (KitKat)

Android versi 4.4 dirilis pada tanggal 31 Oktober 2013 dengan pembaharuan antarmuka, optimasi kinerja pada perangkat dengan spesifikasi rendah, peningkatan tampilan mode layar penuh, dan dukunagn Bluetooth Message Access Profile (MAP).

j. Android versi 5.0 (Lollipop)

Android versi 5.0 dirilis pada tanggal 3 November 2014 dengan perbaharuan antarmuka dengan warna yang lebih hidup. Terdapat beberapa fitur baru, diantaranya penghemat baterai, device sharing, notifikasi, desain materia, dan keamanan yang lebih baik.

2.6 Unity 3D

Unity adalah sebuah ekosistem dari pengembangan *game*, mesin *render* yang terintegrasi penuh dengan satu set peralatan yang intuitif dan alur kerja yang cepat untuk menciptakan konten 3D dan 2D yang interaktif, mudah dipublikasikan ke berbagai *platform*, aset tersedia di *asset store*, dan memiliki komunitas untuk berbagi pengetahuan [7].

Adapun fitur-fitur yang dimilik oleh Unity 3D antara lain sebagai berikut:

a. *Integrated development environment* (IDE) atau lingkungan pengembangan terpadu

b. Penyebaran hasil aplikasi pada banyak *platform*.
c. *Engine* grafis menggunakan Direct3D (Windows), OpenGL (Mac, Windows), OpenGL ES (iOS), and *proprietary* API (Wii).
d. Game *Scripting* melalui Mono. *Scripting* yang dibangun pada Mono, implementasi *open source* dari NET Framework. Selain itu Pemrogram dapat menggunakan UnityScript (bahasa kustom dengan sintaks JavaScriptinspired), bahasa C # atau Boo (yang memiliki sintaks Python-inspired).

Mesh merupakan bentuk dasar dari obyek 3D. Pembuatan *mesh* tidak dilakukan pada Unity. Sementara GameObjects adalah kontainer untuk semua Komponen lainnya. Semua objek dalam permainan disebut *game objects*.

Material digunakan dan dihubungkan dengan *mesh* atau *renderer* partikel yang melekat pada *game object*. Material berhubungan dengan penyaji *Mesh* atau partikel yang melekat pada *GameObject* tersebut. Mereka memainkan bagian penting dalam mendefinisikan bagaimana objek ditampilkan. *Mesh* atau partikel Tidak dapat ditampilkan Tanpa material karena material meliputi referensi untuk *Shader* yang digunakan untuk membuat *Mesh* atau Partikel. *Material* digunakan untuk menempatkan Tekstur ke *GameObjects*.

Unity mendukung pengembangan aplikasi Android. Sebelum dapat menjalankan aplikasi yang dibuat dengan Unity Android diperlukan adanya pengaturan lingkungan pengembang Android pada perangkat. Untuk itu pengembang perlu men-download dan menginstal SDK Android dan menambahkan perangkat fisik ke sistem. Unity Android memungkinkan pemanggilan fungsi kustom yang ditulis dalam C / C + + secara langsung dan Java secara tidak langsung dari script C #.

Perbedaan mendasar antara unity desktop dan unity android adalah sebagai berikut:

1. *Dynamic typing* pada *javascript* tidak diperbolehkan dalam unity android
2. *Terrain Engine* tidak didukung pada perangkat android
3. ETC sebagai *texture compression* dikarenakan android tidak mendukung tekstur DXT/ PVRTC/ ATC.

4. *Movie texture* tidak didukung pada android, tetapi *streaming video* layar penuh disediakan melalui fungsi *scripting*.

2.7 Autodesk 3ds Max

3ds Max merupakan sebuah *software modeling* 3D yang menyediakan model komprehensif, animasi, simulasi, dan *rendering* solusi untuk *game*, film, dan gerak seniman grafis. 3ds Max memberikan alat-alat baru yang efisien, mempercepat kinerja, dan alur kerja yang efisien untuk membantu meningkatkan produktivitas secara keseluruhan untuk bekerja dengan kompleks, aset-resolusi tinggi [8].

2.8 Adobe Photoshop

Adobe Photoshop merupakan program pengolah grafik yang mampu bekerja pada dua tipe grafik yaitu bitmap dan vektor. Oleh sebab itu, file kerja pada Adobe Photoshop dapat berupa gambar bitmap maupun vektor. Hal ini merupakan keunggulan dari program Adobe Photoshop karena dengan kemampuan tersebut akan memudahkan untuk membuat obyek, mengolah foto maupun pengeditan foto lebih lanjut.

2.9 UML

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. [8] UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa pemrograman berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk *modeling* aplikasi prosedural dalam VB atau C.

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek [8].

Melalui berbagai penjelasan rumit yang terdapat di dokumen dan buku-buku UML, sebenarnya konsepsi dasar UML dapat dirangkum seperti pada tabel 2.3

Tabel 2. 3 Konsep Dasar UML

Major Area	***View***	***Diagrams***	***Main Concepts***
Structural	*Static view*	*Class diagram*	*Class, association, generalization, dependency, realization, interface*
	Use case view	*Use case diagram*	*Use case, actor, assosiation, extend, include, use case generalization*
	Implementation view	*Component diagram*	*Component, interface, dependency, realization*
	Deployment view	*Deployment diagram*	*Node, component, dependency, location*
Dynamic	*State machine view*	*Statechart diagram*	*State, event, transition, action*
	Activity view	*Activity diagram*	*State, activity, completion transition, fork, join*
	Interaction view	*Sequence diagram*	*Interaction, object, message, activation*
		Collaboration diagram	*Collaboration, interaction, collaboration role, message*
Model management	*Model management view*	*Class diagram*	*Package, subsystem, model*
Extensibility	*All*	*All*	*Constraint, stereotype, tagged values*

Abstraksi konsep dasar UML yang terdiri dari *structural classification*, *dynamic behavior*, dan *model management*, dapat dipahami dengan mudah apabila melihat tabel 2.3. *Main concepts* dapat dipandang sebagai *term* yang akan muncul pada saat membuat diagram, sedangkan *view* adalah kategori dari diagram tersebut. Berikut merupakan beberapa diagram yang ada pada UML :

a. *Use Case Diagram*
b. *Class Diagram*
c. *Activity Diagram*
d. *Sequence Diagram*

2.9.1 Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, membuat sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor

adalah sebuah *entitas* manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu [8].

Use case diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat menambahkan fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang ditambahkan akan dipanggil setiap kali *use case* yang menambahkan dieksekusi secara normal.

Sebuah *use case* dapat ditambahkan oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat melakukan *extend use case* lain dengan behaviour yang dimiliki. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

2.9.2 Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi) [8].

Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok :

a. Nama (dan *stereotype*)
b. Atribut
c. Metode

Atribut dan metoda dapat memiliki salah satu sifat berikut :

a. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
b. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
c. *Public*, dapat dipanggil oleh siapa saja.

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*. Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*.

Hubungan Antar *Class* :

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas"). Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
3. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence* diagram yang akan dijelaskan kemudian.

2.9.3 Activity Diagram

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity* diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi [8].

Activity diagram merupakan *state* diagram khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity* diagram tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem)

secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari *level* atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan *behaviour* pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity* diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.

2.9.4 Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence* diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait) [8].

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang menjadi *trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan output apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal.

Message digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan icon khusus untuk objek *boundary*, *controller* dan *persistent entity*.

2.10 Star UML

StarUML adalah sebuah proyek *open source* untuk pengembangan secara cepat, fleksibel, *extensible, featureful*, dan bebas-tersedia. UML / platform MDA berjalan pada platform Win32. Tujuan dari proyek StarUML adalah untuk membangun sebuah alat pemodelan perangkat lunak dan juga platform yang menarik adalah pengganti alat UML komersial seperti Rational Rose, Together dan sebagainya.

Star UML mendukung UML (Unified Modeling Language). Berdasarkan pada UML *version* 1.4 dan dilengkapi 11 macam diagram yang berbeda, selanjutnya mendukung notasi UML 2.0 dan juga mendukung pendekatan MDA (*Model Driven Architecture*) dengan dukungan konsep UML. StarUML dapat memaksimalkan pruduktivitas dan kualitas dari suatu software project. UML 2.0 itu sendiri adalah UML standar yang terus berkembang dan dikelola oleh OMG (*Object Management Group*). Baru-baru ini, UML 2,0 direlease dan StarUML dukungan UML 2.0 yang akan mendukung standar terbaru UML.

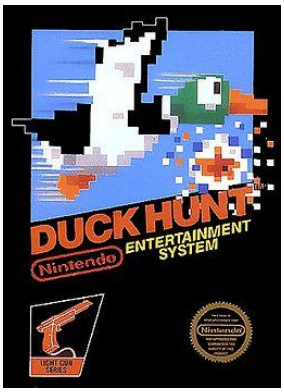
Selanjutnya MDA (Model Driven Architecture). MDA adalah teknologi baru yang diperkenalkan oleh OMG. Untuk mendapatkan keuntungan dari MDA, perangkat lunak alat pemodelan harus mendukung variabel banyak kustomisasi. StarUML dirancang untuk mendukung MDA dan menyediakan kustomisasi banyak variabel seperti profil UML sebagai, Pendekatan, Model Kerangka, NX (ekstensi notasi), kode MDA dan dokumen template dan sebagainya. Ini akan membantu Anda dalam alat pemasangan ke dalam budaya organisasi, proses, dan proyek Anda.

StarUML menyediakan secara sederhana dan kuat arsitektur plug-in sehingga siapapun dapat mengembangkan plug-in modul dalam bahasa COM-kompatibel (C + +, Delphi, C #, VB, dll). Hal yang paling penting dalam pengembangan perangkat lunak adalah *Usability*. StarUML diimplementasikan untuk memberikan berbagai fitur yang *user-friendly* seperti *dialog* Cepat, manipulasi *Keyboard*, ikhtisar Diagram, dll.

StarUML sebagian besar ditulis dalam Delphi. Namun, StarUML adalah proyek multi-bahasa dan tidak terikat dengan bahasa pemrograman tertentu, sehingga setiap bahasa pemrograman dapat digunakan untuk mengembangkan StarUML. (Misalnya, C / C + +, Java, Visual Basic,Delphi, JScript, VBScript, C #, VB.NET, dll).

2.11 Game Duck Hunt

Duck Hunt merupakan *game* yang bertipe *shooting game* yang dimainkan dengan tujuan untuk menembak sasaran yang bergerak pada layar televisi. *Game* ini pertama kali dirilis di Jepang pada tanggal 21 April 1984 oleh Nintendo untuk Nintendo Entertaiment System (NES) video *game* konsol. Cover game duck hunt pada saat diluncurkan untuk konsol Nitendo dapat dilihat pada gambar 2.14



Gambar 2. 14 Cover Game Duck Hunt

Game duck hunt ini dimainkan dari sudut pandang orang pertama (*first person shooter*) dan membutuhkan pistol cahaya NES Zapper, yang mana pemain mengarahkan dan menembak pada layar kaca. Setiap ronde terdiri dari total sepuluh target untuk menembak, tergantung pada mode permainan yang dipilih oleh pemain sebelumnya. Satu atau dua target bebek akan muncul di layar pada waktu tertentu dan pemain memiliki tiga tembakan atau peluru untuk menembak target sebelum mereka menghilang. Desain tampilan game duck hunt untuk versi konsole dapat dilihat pada gambar 2.15



Gambar 2. 15 Tampilan Game Duck Hunt

Seiring dengan perkembangan *game* dalam teknologi *mobile*, game duck hunt mulai dibuat kembali untuk sistem operasi android. Duck hunt versi android tidak jauh berbeda dengan versi *console*, untuk menembak sasaran pemain menggunakan sentuhan pada layar perangkat android. Contoh game duck hunt yang dibuat pada perangkat android dapat dilihat pada gambar 2.16



Gambar 2. 16 Contoh Game Duck Hunt