BAB II

TINJAUAN PUSTAKA

Pada bagian ini akan dijelaskan tentang *state of the art* dari hal-hal mendasar yang berkaitan dengan penelitian ini, yaitu tentang deteksi tabrakan, CVE, dan implementasi deteksi tabrakan pada CVE.

2.1. Deteksi Tabrakan

Deteksi tabrakan (*collision detection*) merupakan bagian yang sangat penting dan mendasar pada banyak aplikasi 3D seperti *computer games*, animasi, simulasi, *virtual prototyping*, robotik, dan sebagainya. *Collision* (tabrakan) adalah suatu konfigurasi dari dua atau beberapa objek yang menempati suatu titik yang sama pada waktu yang sama. Deteksi tabrakan adalah suatu mekanisme yang dapat mendeteksi kapan dan dimana tabrakan terjadi (Bergen, 2004).

Secara detail, deteksi tabrakan berhubungan dengan masalah menentukan **jika**, **kapan**, dan **dimana** dua objek akan melakukan tabrakan. **Jika** berarti menghasilkan nilai *Boolean* yang merupakan jawaban apakah objek-objek akan bertabrakan. **Kapan** berhubungan dengan waktu terjadinya tabrakan, sedangkan **dimana** menyatakan pada titik koordinat (dalam tiga dimensi) berapa tabrakan terjadi (Ericson, 2005).

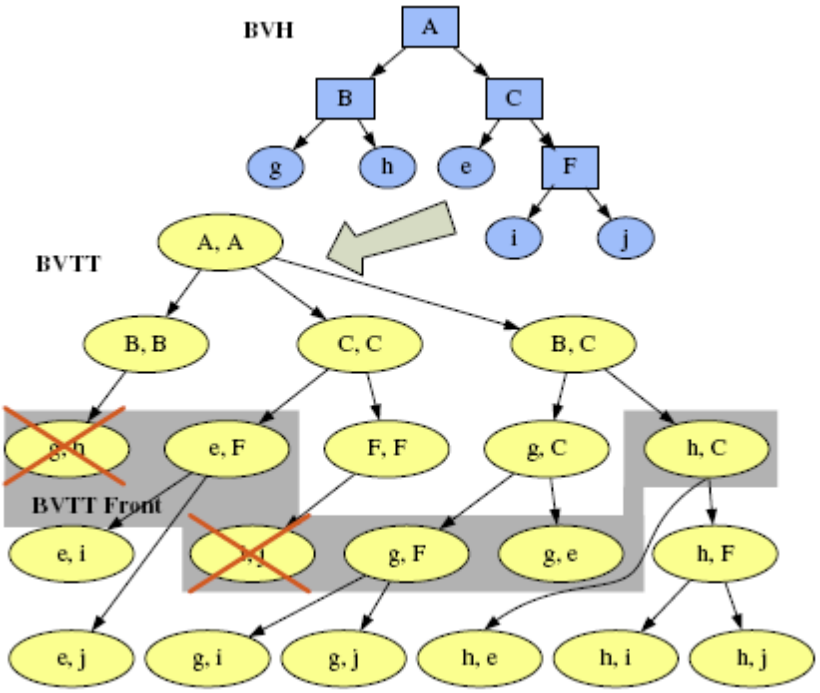
Walaupun kedengarannya sederhana, namun ternyata dalam satu dekade terakhir banyak sekali penelitian yang dilakukan berkaitan dengan deteksi

tabrakan ini. Secara umum, deteksi tabrakan objek ini dapat dikategorikan atas dua: deteksi tabrakan secara diskrit, dan deteksi tabrakan secara kontinu.

Deteksi tabrakan secara diskrit hanya melakukan deteksi tabrakan pada waktu tertentu, misalnya pada waktu *t*. Konsekwensinya adalah metode ini banyak kehilangan pendeteksian tabrakan antara dua konfigurasi yang berurutan. Ini yang disebut oleh para peneliti sebagai *tunneling problem*. Deteksi tabrakan secara diskrit memang tidak memerlukan perhitungan yang banyak sehingga prosesnya lebih cepat. Deteksi tabrakan secara kontinu dapat menghilangkan masalah *tunneling*. Ini disebabkan karena dalam algoritmanya menggunakan gerakan interpolasi untuk memeriksa terjadinya tabrakan dalam gerakan kontinu. Oleh karenanya algoritma ini lebih lambat dibandingkan dengan deteksi tabrakan diskrit (Sulaiman, 2011). Salah satu pendekatan yang digunakan dalam deteksi tabrakan kontinu adalah dengan *bounding volume* yang dapat dilakukan menggunakan *sphere*, *axis aligned bounding box* (AABB) (Bergen, 1997), dan lain-lain.

Karena keandalannya dalam melakukan deteksi tabrakan, metode yang kedua banyak digunakan oleh peneliti dalam menemukan cara untuk membuat metode tersebut menjadi lebih cepat. Beberapa hal yang telah dilakukan adalah dengan pendekatan interpolasi secara linier antara vertex-vertex model dan menghitung waktu pertama terjadinya tabrakan yang didasarkan pada penyeleseksian hirarki (Tang, 2009), dan melakukan pengujian dasar antara pasangan-pasangan segitiga (Curtis, 2008).

Pendekatan lain yang telah dilakukan untuk mempercepat deteksi tabrakan kontinu yaitu menggunakan perhitungan secara parallel dengan memanfaatkan keandalan multi *processor*/CPU (Tang, 2010). Pemrosesan secara parallel ini dilakukan dengan *front-based decomposition* dengan memanfaatkan vertex-vertex yang menjadi *BVTT-front* (lihat Gambar 2.1). Berdasarkan hasil penelitiannya, telah dihasilkan algoritma deteksi tabrakan secara paralel yang dijalankan pada komputer dengan CPU delapan *core* dan komputer dengan 16 *core*. Masing-masing komputer tersebut telah memberikan kecepatan deteksi tabrakan 7x dan 13x lebih cepat.



Gambar 2.1. *Bounding volume hierarchy tree* yang memuat BVTT-front

Selain menggunakan multi *processor*, peningkatan kecepatan deteksi tabrakan telah dilakukan menggunakan beberapa *graphics processing unit* (GPU). Berbeda dengan CPU, prosesor-prosesor GPU sangat sesuai untuk perhitungan secara parallel. Hal ini disebabkan karena jumlah *core* (inti) GPU lebih banyak dibandingkan CPU, juga karena *bandwidth* pada GPU juga lebih tinggi dibandingkan CPU (Lauterbach, 2009).

Karena *bandwidth* antara CPU dan GPU sangat terbatas, maka perlu perpaduan antara CPU dan GPU dalam melakukan perhitungan paralel terhadap deteksi tabrakan. (Kim, 2009) telah menggunakan CPU empat *core* dan dua GPU. Hasil yang diperoleh ternyata lebih cepat 50% sampai dengan 80% dibandingkan dengan hanya menggunakan CPU untuk pengujian pada model yang sama.

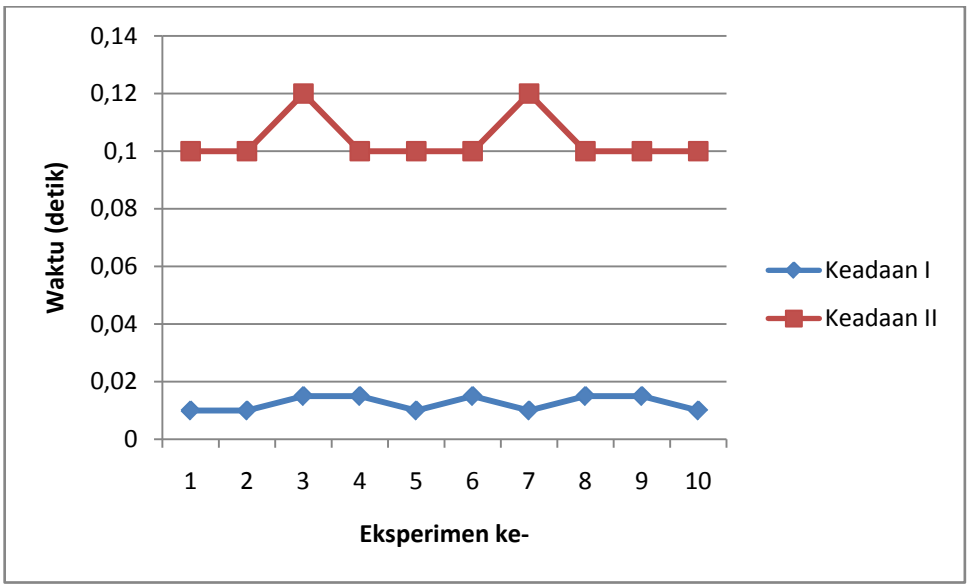
2.2. *Collaborative Virtual Environment*

Virtual Environment (VE) adalah adalah suatu lingkungan sintetis yang cenderung memberikan persepsi bahwa lingkungan dan isinya tersebut seperti tidak sintetis (Bailenson, 2002). Ketika beberapa user dapat saling berinteraksi dalam suatu VE, ini disebut sebagai *collaborative virtual environment* (CVE). Berbagai penelitian terakhir yang berkaitan dengan CVE selalu lebih menitikberatkan pada masalah bagaimana interaksi antar user dapat dilakukan secara *real-time* dan user-user yang terlibat dapat divisualisasikan dalam tiga dimensi (Aspin, 2010). User yang terlibat dalam CVE selain model 3D dari dirinya sendiri, juga dapat berbentuk manusia virtual (*virtual human*) yang sering disebut sebagai avatar (Nassiri, 2009).

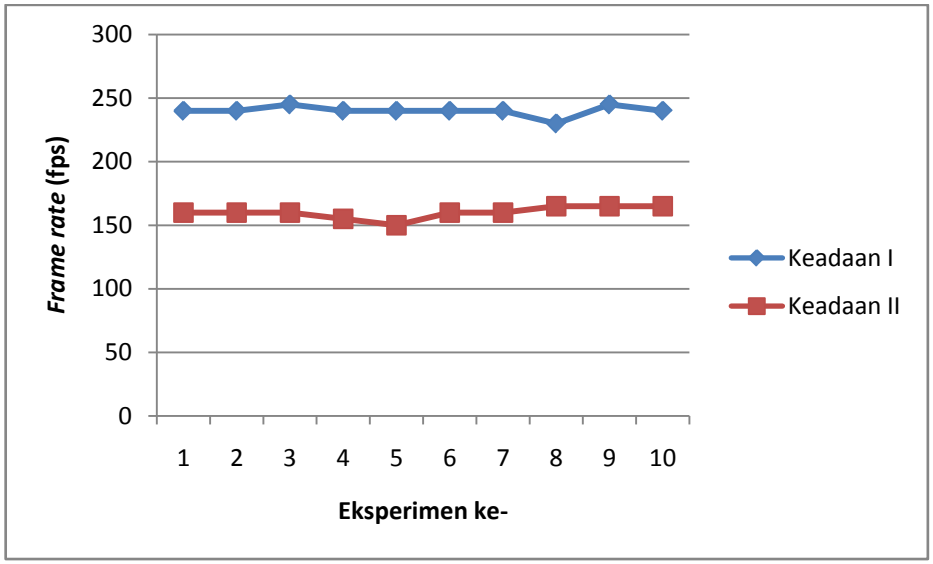
Aplikasi CVE memiliki karakteristik yang selalu meng-*update* lingkungannya. Ini disebabkan karena objek-objek yang terlibat didalamnya selalu berubah, baik sifat maupun kuantitasnya. Untuk itu peningkatan *workload* pada suatu komputasi dalam CVE akan berpengaruh sangat signifikan pada *performance* CVE itu sendiri (Elfizar, 2013a).

2.3. Implementasi Metode Deteksi Tabrakan Objek pada CVE

Metode AABB adalah metode sederhana yang banyak digunakan dalam pendeteksian tabrakan objek pada CVE. Metode ini memiliki *computation cost* yang kecil sehingga alokasi memori untuk pemrosesannya pun kecil. Sayangnya, akurasi metode ini cukup rendah (Elfizar, 2013b).



Gambar 2.2. *Runtime* CVE



Gambar 2.3. *Frame rate* CVE

Penggunaan beberapa metode deteksi tabrakan objek mutakhir seperti OBB, HPCCD, dan MCCD secara langsung pada CVE tidak memungkinkan. Hal ini disebabkan karena seluruh metode tersebut memiliki kekurangan masing-masing. Implementasi metode tersebut pada CVE dengan sendirinya akan menurunkan performa CVE baik dari sisi *runtime* maupun *frame rate* aplikasi (Elfizar, 2013a).

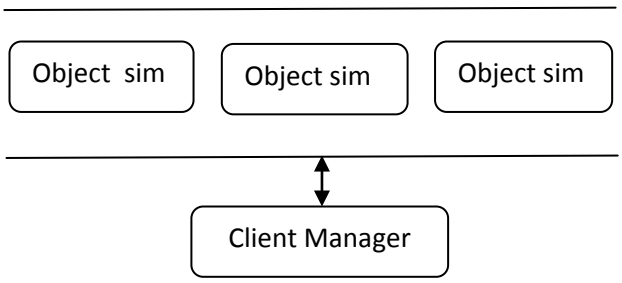
Dari Gambar 2.2 dapat diperhatikan bahwa dengan memberikan tambahan *workload* pada metode AABB (Keadaan II), runtime yang dihasilkan naik secara signifikan bila dibandingkan dengan AABB itu sendiri (Kondisi I). Ini berarti bahwa penambahan *workload* membuat *computation cost* CVE naik secara drastis.

Selanjutnya, bertambahnya *workload* ini juga membuat *frame rate* aplikasi menjadi turun dibandingkan dengan Kondisi I (Gambar 2.3). Ini akan sangat berpengaruh terhadap performa CVE yang digunakan oleh user sebab *frame* yang ditampilkan dari waktu ke waktu menjadi lambat.

Untuk itu diperlukan modifikasi dan penyesuaian pada metode tersebut dan atau mempartisi *workload* dari CVE itu sendiri sehingga dapat dijalankan pada banyak komputer server.

2.4. CVE dengan Simulator Berbasis Objek

Liu (2010) telah menggunakan *Binary Space Partitioning* (BSP) untuk mempartisi beban kerja pada aplikasi CVE untuk mengurangi pekerjaan yang harus ditangani oleh CVE. Namun cara ini tidak ditujukan pada masalah deteksi tabrakan.



Gambar 2.4. Arsitektur simulator berbasis objek

Dengan tujuan yang sama, peningkatan terhadap metode ini telah pun dilakukan menggunakan simulator berbasis objek (*object-based simulators*) (Elfizar, 2012). Dalam simulator berbasis objek, CVE terdiri atas beberapa simulator. Simulator diperuntukkan bagi setiap objek yang ada dalam CVE. Suatu simulator hanya bertanggung jawab menangani tampilan dan tingkah laku suatu objek tertentu. Dengan demikian *workload* simulator tersebut menjadi jauh berkurang. Gambar 2.4 menunjukkan arsitektur CVE yang menggunakan *object-based simulator*.