Pemanfaatan Aljabar Boolean dalam Graphic Processing pada Penglihatan Tembus Dinding di Berbagai Game

Dhafin Fawwaz Ikramullah - 13522084[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika - Komputasi
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*author@itb.ac.id*

***Abstract*—Pada sebuah game, agar pemain tidak kebingungan tentang posisi karakter mereka ketika dibalik dinding, sering kali dinding bagian tersebut dibuat transparan sehingga karakternya dapat dilihat. Salah satu konsep penting untuk melakukan proses grafis tersebut adalah Aljabar Boolean. Aljabar Boolean merupakan cabang aljabar di mana variabel yang ada dapat dimanipulasi dengan nilainya hanya bernilai 1 atau 0. Pada makalah ini akan dibahas sebuah pendekatan memanfaatkan Aljabar Boolean untuk menghasilkan efek dinding tembus pandang tersebut. Proses kalkulasi menggunakan Aljabar Boolean ini terjadi pada proses kalkulasi grafis menggunakan shader. Shader sendiri merupakan sebuah program pada saat melakukan kalkulasi bagaimana menggambarkan sesuatu. Shader merupakan salah satu hal yang bisa dimanfaatkan dalam pemrograman game.**

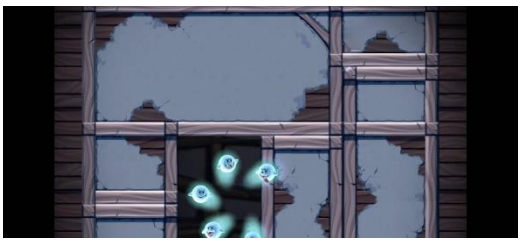
***Kata kunci*— boolean algebra, masking, game, graphic.**

I. PENDAHULUAN

Pada sebuah game, terdapat sebuah solusi dari masalah yang terjadi ketika karakter tertutup oleh dinding. Agar pemain tidak kebingungan tentang posisi karakter yang mereka kendalikan, sering kali kita lihat terdapat efek yaitu beberapa bagian dari dinding yang menutup pemain dijadikan transparan. Salah satu contohnya ada pada gambar di bawah.

Gambar 1. Fallout, Keadaan saat karakter dibelakang dinding. (Sumber:https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRWU3jZZgXljORbTTq0vB2Z9Jtf1ueI41D8hvIXttx66LknxeoT)

Dapat dilihat dari gambar di atas bahwa ketika pemain memasuki area yang akan tertutup, bagian dinding di sekitar pemain dibuat menjadi transparan. Ada pula contoh kasus lain yaitu apa yang akan terjadi jika pemain yang ditutupi dinding ada lebih dari 1. Contoh kasus ini dapat dilihat pada gambar di bawah ini.



Gambar 2. Super Mario Bros U. Keadaan saat tidak ada karakter (Sumber: https://www.youtube.com/watch?v=VZZIXd69ydE&list=PL3vs_m6C8B5ZJ3h6ZD4Lig2pOCZMFVhV5&index=18)

Gambar 3. Super Mario Bros U. Dinding menjadi transparan saat dimasuki karakter (Sumber: https://www.youtube.com/watch?v=VZZIXd69ydE&list=PL3vs_m6C8B5ZJ3h6ZD4Lig2pOCZMFVhV5&index=18)

Jika kita perhatikan, terdapat dua lingkaran yang membuat dinding transparan. Kedua lingkaran ini seakan-akan melakukan sebuah unifikasi antara yang satu dengan yang lain. Pada makalah ini akan dibahas mengenai sebuah pendekatan menggunakan Aljabar Boolean untuk menghasilkan hal ini.

II. LANDASAN TEORI

A. Aljabar Boolean

Aljabar boolean merupakan salah satu cabang dari aljabar

yang hanya menggunakan variabel dengan bilangan biner yang dapat direpresentasikan dengan nilai True atau False [1]. Aljabar Boolean ini ditemukan pada tahun 1854 oleh George Boole. Variabel-variabel yang ada pada Aljabar Boolean dapat diolah dengan berbagai operasi. Operasi aljabar tersebut ada berbaga

Operasi + atau OR
1 + 1 = 1
1 + 0 = 1
0 + 0 = 0

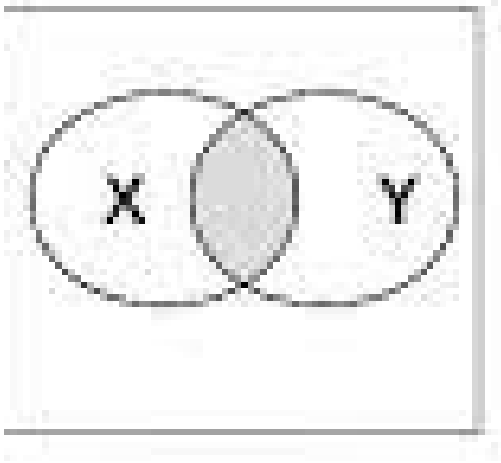
Operasi . atau AND
1 . 1 = 1
1 . 0 = 0
0 . 0 = 0

Ada pula hukum-hukum aljabar 2isband yang juga dapat dimanfaatkan. Diantaranya yaitu:

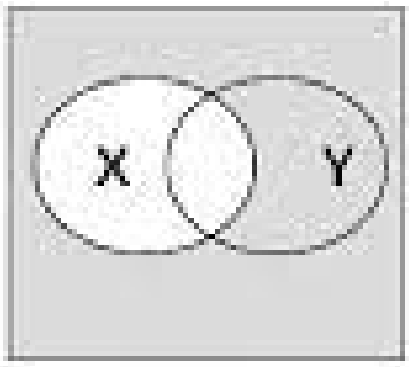
1. Hukum Identitas
   i. a + 0 = a
   ii. a . 1 = a
2. Hukum Indempoten
   i. a + a = a
   ii. a . a = a
3. Hukum Komplemen
   i. a + a' = 1
   ii. aa' = 0
4. Hukum Dominasi
   i. a . 0 = 0
   ii. a + 1 = 1
5. Hukum Involusi
   i. (a')' = a
6. Hukum Penyerapan
   i. a + ab = a
   ii. a (a+b) = a
7. Hukum Komutatif
   i. a + b = b + a
   ii. a . b = b . a
8. Hukum Asosiatif
   i. a + (b + c) = (a + b) + c
   ii. a . (b . c) = (a . b) . c
9. Hukum Distributif
   i. a . (b + c) = a . b + a . c
   ii. a + (b . c) = (a + b) . (a + c)
10. Hukum De Morgan
    i. (a + b)' = a'b' ii. (ab)' = a' + b'
11. Hukum 0/1
    i. 0' = 1
    ii. 1' = 0

Operasi-operasi tadi dapat pula divisualisasikan secara visual dengan sebuah diagram venn. Misalnya

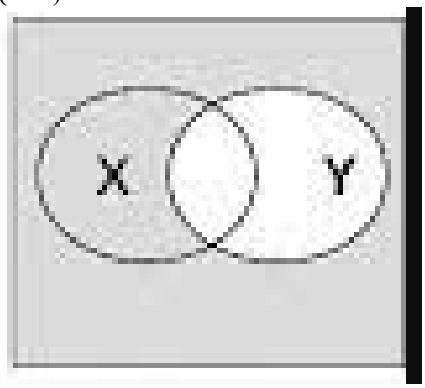
X . Y



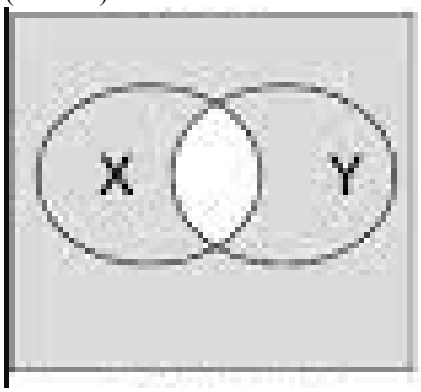
X'



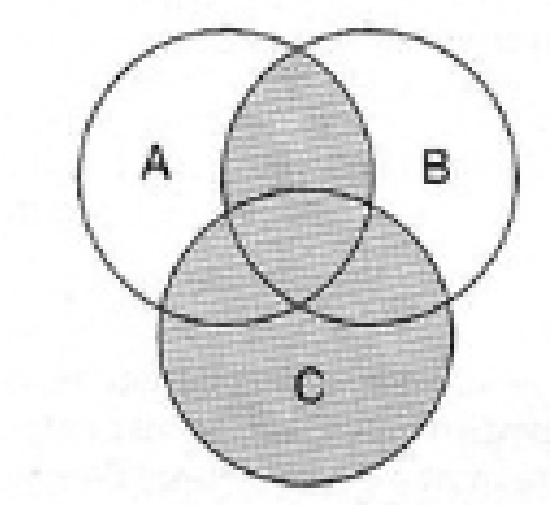
(X.Y)'



(X' + Y')



(A . B) + C



B. Shader

Pada saat sebuah game ingin menggambarkan sesuatu ke layar, ada sebuah program yang akan memberikan instruksi kepada program tentang bagaimana menggambar hal tersebut. Program ini adalah shader [2]. Hampir semua shader dieksekusi dengan GPU (Graphic Processing Unit). Hal hal yang dapat dimodifikasi dengan shader contohnya adalah warna, posisi geometri, transparansi, dan sebagainya. Shader sering digunakan dalam processing visual effect pada sebuah film maupun game.

Mungkin akan muncul pertanyaan, mengapa harus shader. Mengapa tidak dengan programming biasa saja. Bayangkan terdapat ribuan pixel pada layar, masing-masing diproses satu persatu dalam sebuah for loop. Tentu hal ini akan membuat performanya sangat lambat. Apalagi mesin dituntut mengulang proses ini setiap 1/30 detik untun mencapai 30 frame per second. Karena hak inilah digunakan shader. Shader dibuat untuk mengaplikasikan sebuah transformasi dari lata yang sangat besar dalam waktu bersamaan secara serentak. Contohnya, dari pada melakukan for loop untuk memproses masing-masing pixel pada layar secara sekuensial, dengan shader kita memproses masing-masing pixel tersebut secara bersamaan atau serentak. GPU sangat baik dalam melakukan sebuah proses seperti contoh kasus tadi CPU.

Bahasa pemrograman yang dapat digunakan untuk shader contihnya adalah GLSL (OpenGL Shading Language). Untuk Direct3D yaitu HLSL (High Level Shader Language) yang dibuat oleh Nvidia.

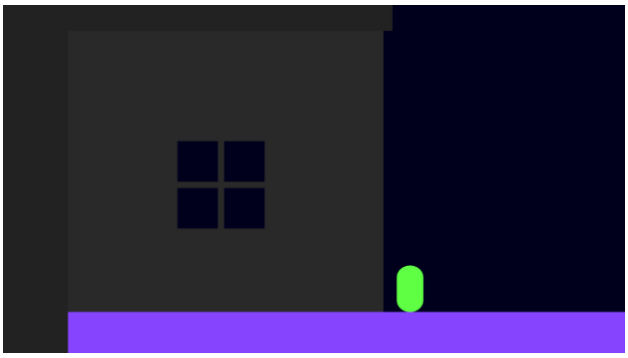
Dalam aplikasinya, dengan shader kita dapat menghasilkan sebuah grafis yang sangat bervariasi seperti contoh gambar berikut.

Gambar 4. Aplikasi shader untuk pembuatan visual effect fractured cube

III. PEMBAHASAN

A. Setup Awal

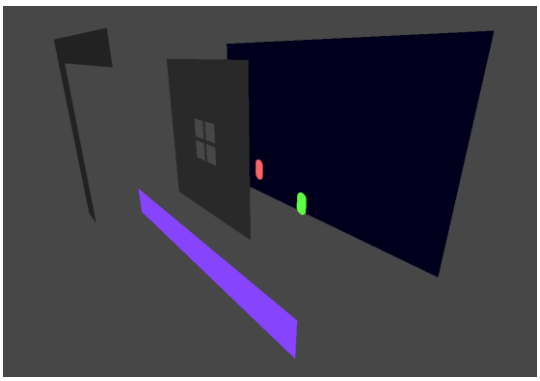
Dalam implementasi program, akan dimanfaatkan framework/engine yang sudah tersedia yaitu akan digunakan shader pada Unity Engine untuk mempermudah proses. Kemudian dibuat sebuah layout dari sebuah level game secara primitif. Dapat dilihat pada gambar sebagai berikut.



Agar lebih sederhana, dapat dilihat pada gambar terdapat beberapa objek untuk merepresentasikan objek objek sebenarnya yaitu:

- Kapsul: karakter yang dapat digerakkan pemain
- kotak abu-abu tua: dinding samping yang tidak bisa dilewati pemain
- ungu: lantai untun pijakan pemain
- kotak abu-abu muda: dinding depan yang dapat menutupi pemain sehingga pemain tidak terlihat.
- kotak biru tua di belakang: sebagai background.

Masing-masing objek ini akan disusun seperti memiliki layernya tersendiri untuk mempermudah penjelasan. Hal ini dapat dilihat pada gambar sebagai berikut.



Masing-masing object telah disusun berdasarkan jenisnya masing-masing. Urutannya dari yang terdepan yaitu dinding penutup karakter, lantai dan dinding samping, karakter, background. Untuk mempermudah pengetesan, kedua karakter juga bisa ditarik menggunakan mouse.

B. Perancangan Kode

Terdapat beberapa informasi yang dibutuhkan oleh shader yaitu hal-hak seperti resolusi layar, posisi kamera, dan sebagainya. Hal-hak ini sudah disediakan oleh engine sehingga dapat langsung digunakan. Tetapi terdapat informasi yang harus kita pass ke shader sendiri. Informasi tersebut adalah posisi kedua karakter. Maka sebelum melakukan implementasi shader, kita pass dulu data posisi pemain ke shader. Hal ini dapat dicapai dengan sangat sederhana yaitu sebagai berikut.

```
0 references
public class GameManager : MonoBehaviour
{
    2 references
    [SerializeField] Transform _player1Trans;
    2 references
    [SerializeField] Transform _player2Trans;
    3 references
    [SerializeField] Material _hiddenAreaMat;

    0 references
    void Update()
    {
        UpdateShader();
    }
    2 references
    void UpdateShader()
    {
        // Pass player position to shader
        _hiddenAreaMat.SetVector("_Position1", _player1Trans.position);
        _hiddenAreaMat.SetVector("_Position2", _player2Trans.position);
    }
```

Gambar 5. Kode untuk passing data posisi karakter ke shader

Sekarang akan dimulai implementasi dari shader tersebut. Karena kurang relevan dari pembahasan utama makalah ini, maka implementasi sampai bagian aljabar boolean akan dijelaskan dengan sangat singkat. Dapat dilihat pada penggalan code berikut.

```
Shader "Custom/WallCover" {
    Properties {
        [NoScaleOffset]_MainTex ("_MainTex", 2D) = "white" {}
        _Radius ("Radius", Float) = 0.3
        _Position1 ("Position1", Vector) = (0, 0, 0, 0)
        _Position2 ("Position2", Vector) = (0, 0, 0, 0)
        _Offset ("Offset", Vector) = (0, -0.2, 0, 0)
    }
    SubShader {
        Tags {
            "RenderType"="Transparent"
            "Queue"="Transparent"
        }

        Pass {
            Blend SrcAlpha OneMinusSrcAlpha
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"

            struct appdata {
                float4 vertex : POSITION;
                float2 uv : TEXCOORD0;
                float4 color : COLOR;
            };

            struct v2f {
                float2 uv : TEXCOORD0;
                float4 vertex : SV_POSITION;
                float4 color : COLOR;
                float4 screenPos : TEXCOORD1;
            };

            sampler2D _MainTex;
            float4 _MainTex_ST;
            float _Radius;
            float4 _Position1;
            float4 _Position2;
            float4 _Offset;
```

Gambar 6. Implementasi shader untuk 1 karakter bagian pertama.

```
            void Ellipse(float2 UV, float Width, float Height, out float4 Out)
            {
                float d = length((UV * 2 - 1) / float2(Width, Height));
                Out = saturate((1 - d) / fwidth(d));
            }

            void Get_Screen_UV(float3 _Position, float3 _Offset, float4 screenPos, out float2 UV)
            {
                float3 circleScreenPos = _Position + _Offset - _WorldSpaceCameraPos;
                float2 uvOffset = float2(circleScreenPos.x/unity_OrthoParams.x,
                    circleScreenPos.y/unity_OrthoParams.y) * -0.5;
                UV = screenPos.xy + uvOffset;
            }

            v2f vert (appdata v) {
                v2f o;
                o.vertex = UnityObjectToClipPos(v.vertex);
                o.uv = v.uv;
                o.color = v.color;
                o.screenPos = ComputeScreenPos(o.vertex);
                return o;
            }

            fixed4 frag (v2f i) : SV_Target {
                fixed4 col = tex2D(_MainTex, i.uv) * i.color;

                float2 UV1;
                Get_Screen_UV(_Position1, _Offset, i.screenPos, UV1);

                float width = (_ScreenParams.y / _ScreenParams.x) * _Radius;

                // Normalnya float4 untuk alpha (transparansi)
                // tapi karena pembahasannya aljabar boolean jadi langsung bool4 aja
                // Nilainya 0 atau 1 (true atau false)
                bool4 a1, wall;

                wall = col.a;
                Ellipse(UV1, width, _Radius, a1);

                col.a = !a1 & wall;

                return col;
            }
            ENDCG
        }
    }
}
```

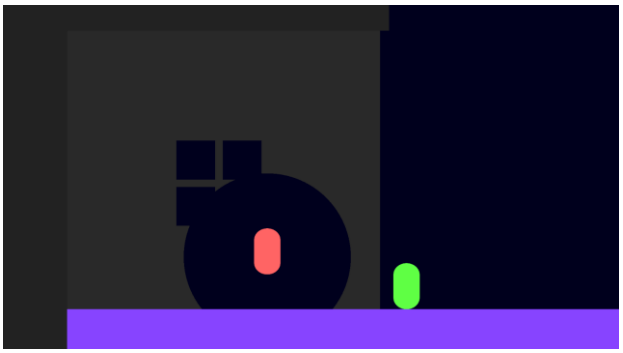
Gambar 7. Implementasi shader untuk 1 karakter bagian kedua.

Secara singkat, yang terjadi pada kode tersebut yaitu, hitung jarak dari posisi kamera ke karakter, gunakan hasilnya sebagai posisi relatif karakter terhadap layar. Gambarkan sebuah lingkaran dengan isi dalam lingkaran bernilai 1, sisanya 0. Posisikan lingkaran ini pada layar sesuai dengan posisi relatif karakter tadi. Maka saat ini variabel a1 sudah memiliki

informasi yaitu sebuah lingkaran yang posisinya sama dengan posisi karakter pada layar, bernilai 1 dalam lingkaran, 0 di luar lingkaran. Sedangkan untuk layar itu sendiri ada pada variabel wall yang mengandung informasi transparansi dari dinding tersebut yaitu semuanya bernilai 1. Dari kedua variabel ini kita dapat memanfaatkan aljabar boolean untuk dimanipulasi.

Ingat kembali pada pembahasan aljabar boolean bahwa jika kita melakukan operasi negasi, lingkaran tadi yang awalnya bernilai 1 di dalam dan 0 diluar akan menjadi berbalik sehingga bernilai 0 di dalam dan 1 diluar.

Lalu dengan memanfaatkan operasi AND antara transparansi dinding yang nilainya 1 semua, dengan lingkaran tadi yang sudah dinegasi, maka kita akan mendapatkan dinding yang memiliki lubang lingkaran. Hal ini karena dengan operasi AND, setiap angka 1 bertemu 1 (bagian luar lingkaran setelah dinegasi) akan tetap menjadi 1, sedangkan angka 1 bertemu 0 (isi lingkaran yang telah dinegasi) akan menjadi 0. Mungkin muncul pertanyaan mengapa tidak langsung set transparansinya dengan lingkaran tadi saja. Ini karena dindingnya tidak selalu berbentuk kotak dan transparansinya bernilai 1 semua. Bisa saja ada bagian yang transparan. Hal ini dapat divisualisasikan lebih mudah dengan gambar berikut.



Gambar 8. Gambar hasil kalkulasi untuk 1 karakter

Kita sudah menyelesaikan untuk karakter pertama. Sekarang lakukan kembali untuk karakter kedua. Kode shader akan menjadi sebagai berikut.

```
63
64      fixed4 frag (v2f i) : SV_Target {
65          fixed4 col = tex2D(_MainTex, i.uv) * i.color;
66
67          float2 UV1, UV2;
68          Get_Screen_UV(_Position1, _Offset, i.screenPos, UV1);
69          Get_Screen_UV(_Position2, _Offset, i.screenPos, UV2);
70
71          float width = (_ScreenParams.y / _ScreenParams.x) * _Radius;
72
73          // Normalnya float4 untuk alpha (transparansi)
74          // tapi karena pembahasannya aljabar boolean jadi langsung bool4 aja
75          // Nilainya 0 atau 1 (true atau false)
76          bool4 a1, a2, wall;
77
78          wall = col.a;
79          Ellipse(UV1, width, _Radius, a1);
80          Ellipse(UV2, width, _Radius, a2);
81
82          col.a = !a1 & !a2 & wall;
83
84          return col;
85      }
86      ENDCG
```

Gambar 9. Kode shader setelah ditambahkan untuk karakter kedua

Variabel a2 akan mengandung informasi lingkaran yang kedua yaitu untuk lingkaran pada pemain ke dua. Maka yang perlu dilakukan sekarang adalah bagaimana cara melakukan unifikasi antara kedua lingkaran tersebut. Tentunya terdapat banyak cara untuk menyelesaikan hal ini. Namun salah satu cara yang mudah dan efisien adalah dengan mengulangi langkah untuk lingkaran pertama yaitu kita cari negasi untuk lingkaran baru. Kemudian lakukan operasi AND kepada kedua lingkaran yang kita punya. Maka akan dihasilkan union dari hasil negasi kedua lingkaran. Dapat divisualisasikan sebagai berikut.

Gambar 10. Gambar hasil aljabar boolean untuk kedua karaker dan dinding di depannya

Kedua lingkaran ini seakan akan merupakan sebuah “mask” untuk dinding depan. Maka dari operasi tadi, kita dapat sederhanakan dengan aljabar boolean. Misalkan

- a1 = karakter 1
- a2 = karakter 2
- wall = dinding depan

Maka rumusnya adalah

(NOT a1) AND (NOT a2) AND wall

V. KESIMPULAN

Berbagai proses kalkulasi grafis pada game dapat diselesaikan memanfaatkan konsep aljabar boolean.

VI. LAMPIRAN

Source code dari pembahasan makalah ini:
https://github.com/DhafinFawwaz/Boolean-Algebra-In-Shader

VII. UCAPAN TERIMA KASIH

Pertama-tama terima kasih kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya penulis dapat menyelesaikan makalah ini dengan baik dan selesai pada waktunya. Tentunya penulis juga berterima kasih kepada keluarga dan teman-teman yang memberikan dukungan sehingga penulis dapat menyelesaikan makalah ini.

REFERENCES

[1] Gonzales, Patricio. 2023. “The Book of Shaders: What is a shader”. https://thebookofshaders.com/01/.

[2] Munir, Rinaldi. 2023. “Aljabar Boolean (Bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 7 Desember 2023

Dhafin Fawwaz Ikramullah, 13522084