

**Laporan Tugas Besar 3**  
**IF3260 Grafika Komputer**  
**Articulated Model**



Disusun oleh:

Felix Setiawan (13518078)

Yonatan Viody (13518120)

Stephen Thajeb (13518150)

**Program Studi Teknik Informatika**  
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2021**

## **DAFTAR ISI**

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB I</b>	
<b>DESKRIPSI</b>	<b>3</b>
<b>BAB II</b>	
<b>HASIL PENGUJIAN</b>	<b>11</b>
<b>BAB III</b>	
<b>CONTOH FUNGSIONALITAS PROGRAM</b>	<b>22</b>
<b>BAB IV</b>	
<b>PEMBAGIAN KERJA</b>	<b>24</b>
<b>REFERENSI</b>	<b>25</b>

## BAB I

### DESKRIPSI

Pada tugas kali ini, kami diminta untuk merancang *articulated model* berbasis WebGL. Terdapat 3 jenis *articulated model* yang telah kami rancang, yaitu:

#### 1. Minecraft Pig

Minecraft Pig adalah *articulated model* yang dibuat berdasarkan entitas pig di permainan Minecraft. Untuk model ini, kami menggunakan *texture mapping* berupa gambar skin minecraft pig. Struktur model ini berupa:

```
| body  
| head  
| nose  
| right-arm (kaki depan)  
| left-arm (kaki depan)  
| right-leg  
| left-leg
```

Berikut adalah tampilan dari *articulated model* Minecraft Pig:



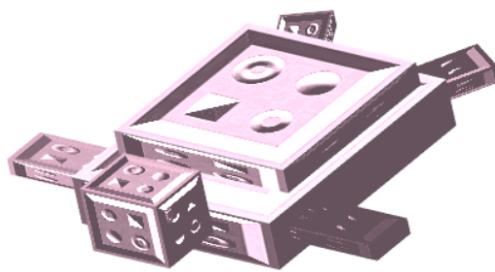
#### 2. Minecraft Turtle

Minecraft Turtle adalah *articulated model* yang dibuat berdasarkan entitas turtle di permainan Minecraft. Untuk model ini, kami menggunakan *bump mapping* berupa gambar yang didapatkan dari website [approvaj.io](http://approvaj.io). Struktur model ini berupa:

```
| body  
| head  
| right-eye  
| left-eye  
| shell  
| right-arm (kaki depan)
```

```
| left-arm (kaki depan)  
| right-leg  
| left-leg
```

Berikut adalah tampilan dari *articulated model* Minecraft Turtle:



### 3. R3D3

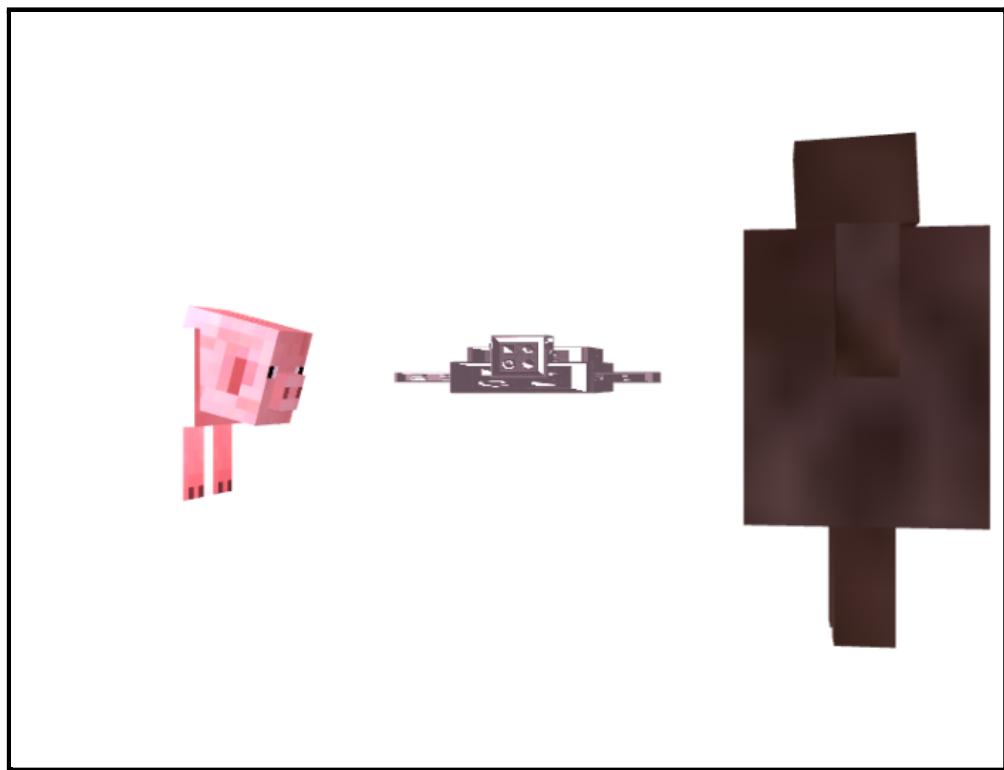
R3D3 adalah articulated model berupa robot yang memiliki struktur fisik menyerupai manusia dan memiliki struktur model yang terdiri dari *head*, *body*, *left arm*, *right arm*, *left leg*, *right leg*. Untuk model ini, kami menggunakan *environment mapping* berupa gambar tembok batu bata. Struktur model ini berupa:

```
| body  
| head  
| right-arm  
| left-arm  
| right-leg  
| left-leg
```

Berikut side view dari *articulated model* R3D3:



Tampak gabungan model:



Kita juga telah merancang program dalam bentuk website yang dapat menerima interaksi dari user untuk melakukan aksi-aksi berikut, yaitu:

1. Membuka file model dari *articulated model* yang telah dibuat.
2. Mengubah jenis proyeksi menjadi *orthographic/oblique/perspective* dan mengubah parameternya.
3. Me-reset ke *default view* (tidak ada proyeksi dan *model matrix* bernilai matriks identitas).
4. Melakukan transformasi bagian-bagian model yang dipilih sesuai dengan sendi yang tersedia.
5. Mengubah jarak *camera view* untuk melakukan *zoom-in / zoom-out*.
6. Menggerakkan kamera untuk mengitari *world view*.
7. Melakukan *shading* dan *texture mapping* per bagian model yang dapat dimatikan/dinyalakan.
8. Menjalankan dan menghentikan skenario animasi dari semua model pada *canvas*.

Untuk membuat kode program yang lebih terstruktur dan mudah dibaca, kami menuliskan program dengan pendekatan **Object Oriented**. Secara garis besar, terdapat beberapa kelas yang kami buat untuk program ini, antara lain:

- a. **Kelas MainView** merupakan kelas yang berhubungan langsung dengan interaksi pada *canvas*. MainView akan melakukan inisiasi WebGL pada *canvas* beserta variabel-variabel seperti matriks transformasi dan *lighting*. Selain itu, MainView akan mengolah input user berupa *hold-and-drag* (*mouse click* dan *mouse down*) untuk melakukan rotasi terhadap *world* (model).
- b. **Kelas Observer** merupakan kelas yang menjadi *controller* dan pengamatan atas interaksi dari user. Observer akan melakukan inisiasi *articulated models*, serta UI berupa input dan button untuk melakukan interaksi seperti melakukan transformasi objek (termasuk *shading* dan *texture*), pengubahan proyeksi, serta pengubahan keadaan kamera. Observer juga akan melakukan penggambaran objek ke **MainView** menggunakan fungsi **drawObjects** yang juga digunakan untuk mensimulasikan proyeksi, kamera, dan transformasi model. Observer juga dapat menampilkan animasi model ke **MainView** menggunakan fungsi **animateObjects** yang sebenarnya melakukan **drawObjects**, namun memanfaatkan *keyframes* dari model untuk menentukan transformasinya.
- c. **Kelas Object** merupakan kelas abstrak yang merepresentasi sebuah objek yang dapat ditransformasi. **Object** mengimplementasikan struktur *left-child-right-sibling*. **Object** menggunakan *builder pattern* dalam pembentukan matriks transformasinya.

Selain itu, **Object** dapat mendefinisikan sendi-sendi melalui atribut **bound** dan dapat mendefinisikan *animation keyframes*. Atribut **bound** mendefinisikan apakah sebuah

objek dapat melakukan rotasi dan translasi, termasuk juga sumbu transformasinya. *Animation keyframes* didefinisikan menggunakan struktur *array of frames* dengan frame berbentuk *[frame\_number, state]* dan state berisi sudut rotasi dan besar translasi. Keyframes akan diurutkan berdasarkan *frame\_number* dari kecil ke besar.

Implementasi left-child-right-sibling dilakukan dengan mendefinisikan sebuah atribut *child* dan *sibling*. Untuk mempermudah penambahan child atau sibling, kami mengimplementasi 2 method, yaitu

- `addSibling(obj: Object)` - menambahkan sebuah objek sebagai *sibling* dari objek ini dengan cara melakukan iterasi hingga mencapai *last sibling* atau dirinya sendiri, kemudian men-set atribut *sibling* pada objek hasil iterasi tersebut dengan objek baru.
- `addChild(obj: Object)` - menambahkan sebuah objek sebagai *child* dari objek ini dengan cara melakukan iterasi hingga mencapai *last child* (konsepnya sama dengan melakukan iterasi *sibling*), kemudian men-set atribut *sibling* pada objek hasil iterasi tersebut dengan objek baru. Namun jika objek target pada mulanya belum memiliki *child*, objek baru di-set sebagai atribut *child* pada objek target tersebut.

- d. **Kelas Polygon** merupakan kelas yang merepresentasikan objek poligon dan merupakan **Object**. **Polygon** menyimpan informasi koordinat simpul, warna, koefisien *shininess*, vektor normal, koordinat *texture*, dan vektor *tangent*.
- e. **Kelas Cube** merupakan kelas yang merepresentasikan objek kubus dan merupakan **Object**. **Cube** dapat melakukan generasi objek kubus dan *parsing* objek menjadi sebuah model JSON yang dapat disimpan dalam file. **Cube** juga dapat dibentuk kembali dari data model yang telah tersimpan.
- f. **Kelas MinecraftPigModel** merupakan kelas yang merepresentasikan *articulated model* Minecraft Pig dan merupakan **Object**. **MinecraftPigModel** dapat melakukan generasi objek kubus dan parsing objek menjadi sebuah model JSON yang dapat disimpan dalam file. **MinecraftPigModel** juga dapat dibentuk kembali dari data model yang telah tersimpan. Model ini dibentuk menggunakan beberapa kubus yang ditransformasi sedemikian rupa membentuk model yang diinginkan. Kemudian setiap kubus dihubungkan menggunakan representasi *linked list left-child-right-sibling*. **MinecraftPigModel** menggunakan *texture mapping* berupa gambar/pola.

- g. **Kelas MinecraftTurtleModel** merupakan kelas yang merepresentasikan *articulated model* Minecraft Turtle dan merupakan **Object**. **MinecraftTurtleModel** memiliki struktur yang mirip seperti model **MinecraftPigModel**, namun menggunakan *texture mapping* yang berbeda, yaitu *bump mapping*.
- h. **Kelas R3D3** merupakan kelas yang merepresentasikan *articulated model* Robot R3D3. Sama seperti 2 model sebelumnya, model ini merupakan turunan kelas dari Class **GeoObject** dan kelas ini juga memiliki struktur yang menyerupai class **MinecraftTurtle** dan **MinecraftPigModel**, namun pada model R3D3 menggunakan *environment mapping*.

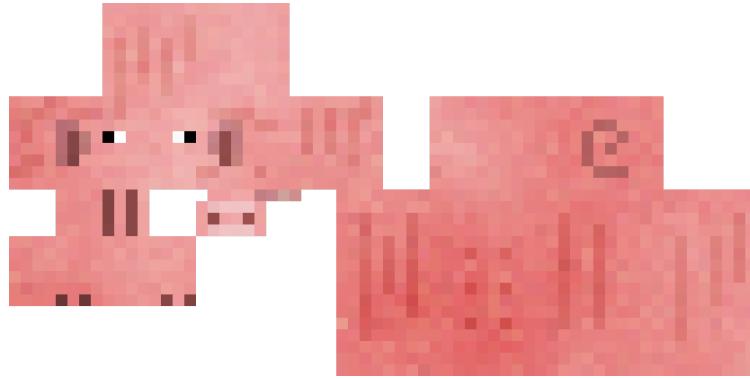
Program yang kami buat telah menyediakan section-section untuk mengatur proyeksi, melakukan transformasi bagian-bagian model (termasuk *shading* dan *texture*), dan melakukan transformasi kamera. Setiap pengaturan ini terhubung langsung pada **Observer** yang akan berinteraksi dengan **MainView** untuk melakukan penggambaran *world* dari hasil pengaturan tersebut. Implementasi matriks transformasi dan proyeksi menggunakan referensi di bawah.

Untuk melakukan *shading*, kami menerapkan model phong yang terdiri dari *diffuse*, *specular*, dan *ambient lighting*. Untuk melakukan perhitungan *diffuse* dan *specular lighting*, kami mendefinisikan vektor normal untuk setiap poligon pada objek yang kami buat. Selain itu, kami juga mendefinisikan posisi pengamat dan light pada [-500, 500, -10] atau *left-top* sehingga sisi yang mengarah ke kiri atas lebih bersinar dibanding sisi yang lain.

Untuk melakukan *texture mapping*, kami menggunakan sebuah variabel yang menentukan tekstur mana yang akan digunakan berupa *textures* dan diinisialisasi pada Observer. Pada program kami, kami mengimplementasi 3 *texture mapping*, yaitu:

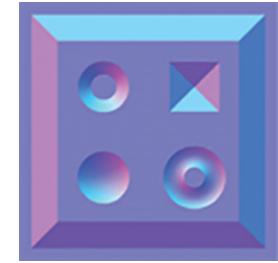
1. Image/Pattern

Untuk menampilkan gambar pada objek yang ada pada canvas, kita perlu mendefinisikan sebuah gambar dan *texture coordinates* (dalam dimensi 2D u dan v) untuk setiap vertex. Gambar dapat di-load menggunakan DOM Image dan di-load memanfaatkan `gl.TexImage2D`. Kemudian kami melakukan *wrapping* `CLAMP_TO_EDGE` jika ukuran gambar tidak berupa hasil perpangkatan dari dua. Gambar yang digunakan untuk mapping ini dapat dilihat di bawah:



## 2. Bump

Pada tugas ini, kami menggunakan *normal bump mapping*. Untuk melakukan *normal bump mapping*, kita melakukan pertubasi terhadap vektor normal. Pertubasi dapat dilakukan menggunakan sebuah gambar berisi vektor normal dengan berbasis *tangent space* (contohnya dapat dilihat pada gambar di samping).



Akibatnya setiap poligon perlu didefinisikan vektor *tangent*-nya dan kalkulasi dalam model phong perlu dilakukan dalam *tangent space* sehingga vektor ke cahaya, vektor ke viewer, dan vektor ke titik *fragment* perlu diubah ke dalam basis *tangent space*. Untuk membuat matrix transformasi dari *world space* ke *tangent space*, kami mendefinisikan *tangent* dan *normal* untuk setiap poligon, kemudian melakukan *cross product* untuk menghasilkan *bitangent* dengan rumus  $B = N \times T$ . Kemudian matrix transformasi basis dapat dibentuk sebagai berikut:

$$TBN = \begin{bmatrix} Tx & Bx & Nx \\ Ty & By & Ny \\ Tz & Bz & Nz \end{bmatrix}$$

Kemudian kita lakukan transformasi basis terhadap parameter model phong sebagai berikut:

- $toLight_{tangent-space} = TBN \cdot toLight$
- $toViewer_{tangent-space} = TBN \cdot toViewer$
- $eyePos_{tangent-space} = TBN \cdot eyePos$

Setelah itu, kita dapat meload gambar yang berisi vektor normal dalam *tangent space* tersebut, dengan cara yang sama seperti sebelumnya, untuk digunakan sebagai vektor normal dalam kalkulasi model phong-blinn.

### 3. Environment

Environment mapping dilakukan dengan menggunakan *intermediate mapping*, di mana kita membuat sebuah bentuk kubus yang berisi tekstur gambar lingkungan. Dalam program kami, kami menggunakan gambar tembok batu bata untuk mensimulasikan lingkungan (di-load pada setiap sisi kubus). Tekstur kubus di-load ke GPU menggunakan gl.TEXTURE\_CUBE\_MAP dan setiap sisi kubus akan di-load dengan cara yang sama seperti pada 2D, yaitu menggunakan gl.texImage2D dengan target gl.TEXTURE\_CUBE\_MAP\_POSITIVE\_X, gl.TEXTURE\_CUBE\_MAP\_NEGATIVE\_X, dll. Tekstur ini akan di-load pada *texture unit* 1 karena unit 0 sudah digunakan untuk texture 2D.



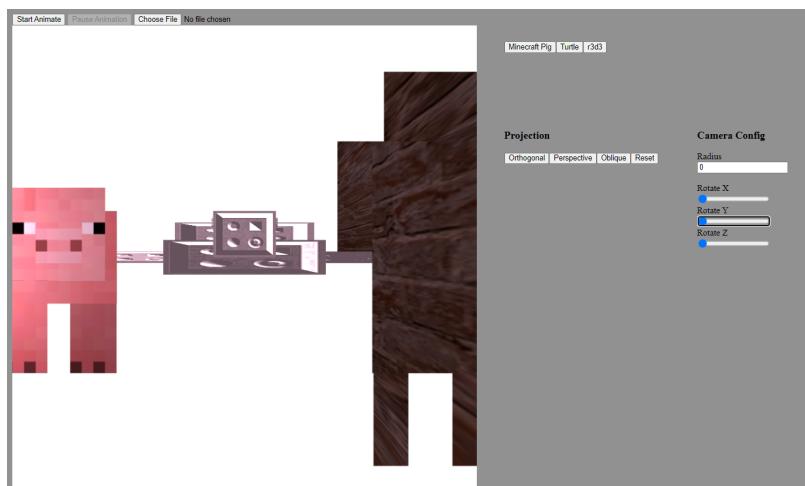
Tekstur kubus ini kemudian di-mapping ke objek dengan menggunakan refleksi antara vektor ke titik *fragment (to viewer)* dengan vektor normal. Akibatnya seakan-akan ada cahaya (sesuai tekstur yang mengenai vektor hasil refleksi) yang datang dan dipantulkan ke arah kita dari titik *fragment* tersebut.

## BAB II

### HASIL PENGUJIAN

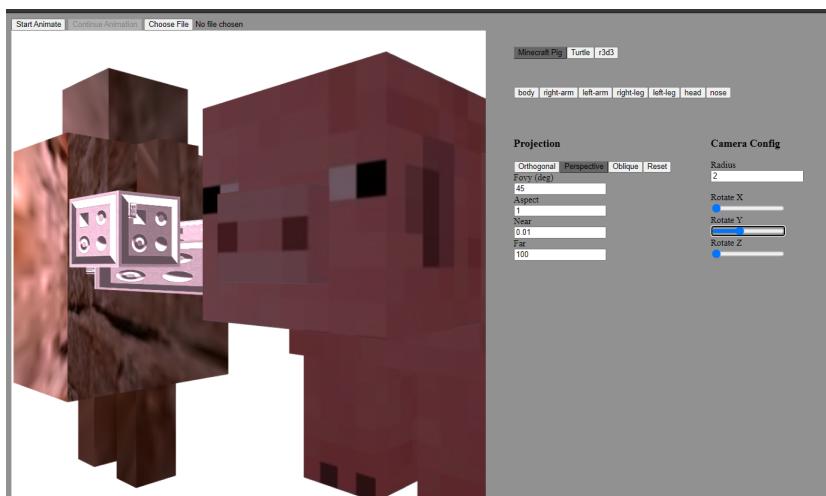
Berikut merupakan hasil pengujian dari program kami, antara lain:

1. Mengubah proyeksi menjadi:
  - a. Orthogonal



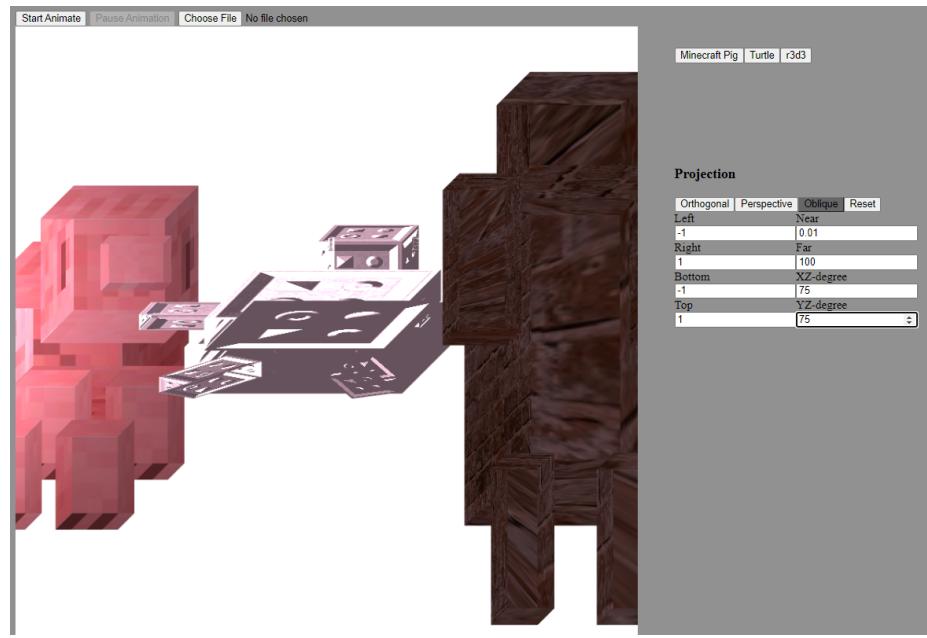
Proyeksi di atas dilakukan dengan parameter *default orthogonal* pada model Pig dan Turtle. Dapat dilihat pada beberapa sisi bagian model, sisi menjadi tembus pandang karena koordinat z di-set menjadi 0.

- b. Perspective



Proyeksi di atas dilakukan dengan parameter *default perspective* dan *camera config* dengan rotate Y sekitar 170 degree dan *radius* bernilai 2, pada model Pig dan Turtle. Dapat dilihat model seakan-akan tampak seperti di depan kita (dalam perspektif pengamat).

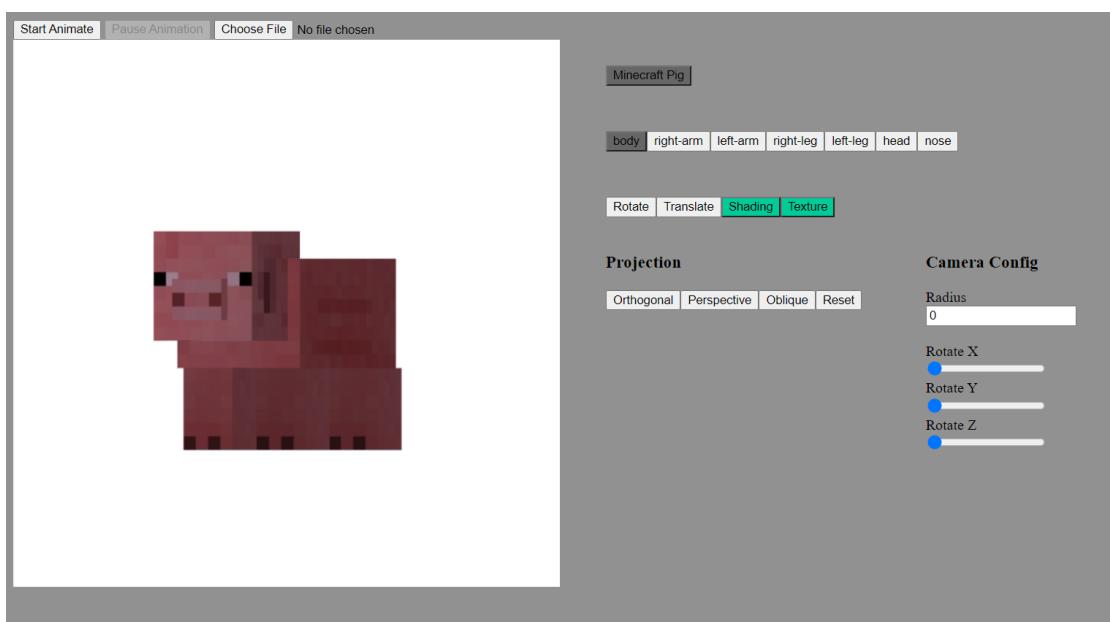
- c. Oblique



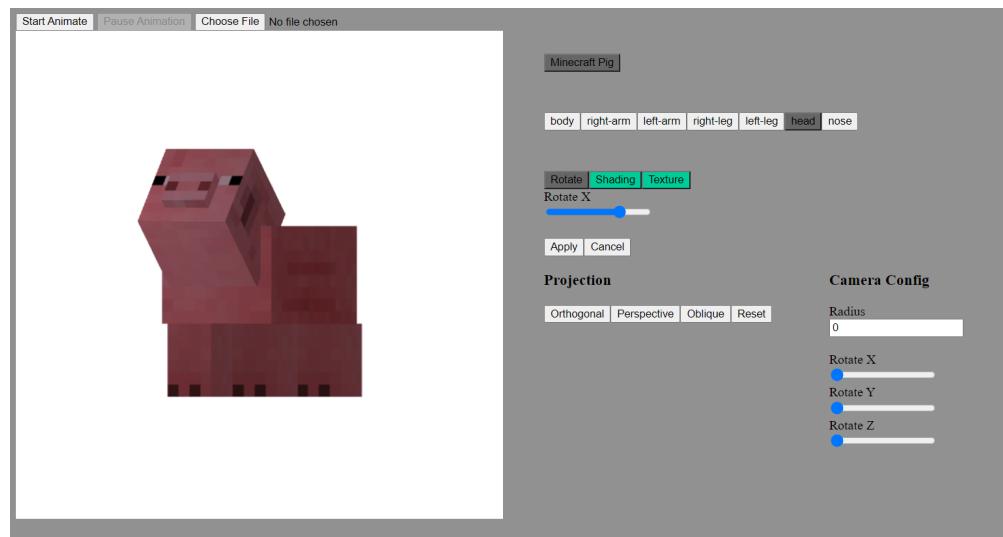
Proyeksi di atas dilakukan dengan parameter *default oblique*, pada model Pig dan Turtle. Dapat dilihat bagian-bagian model berupa kubus menjadi tampak seperti kubus yang di-*shear*.

2. Melakukan rotasi dan translasi dari bagian-bagian model yang dipilih:

Gambar awal:

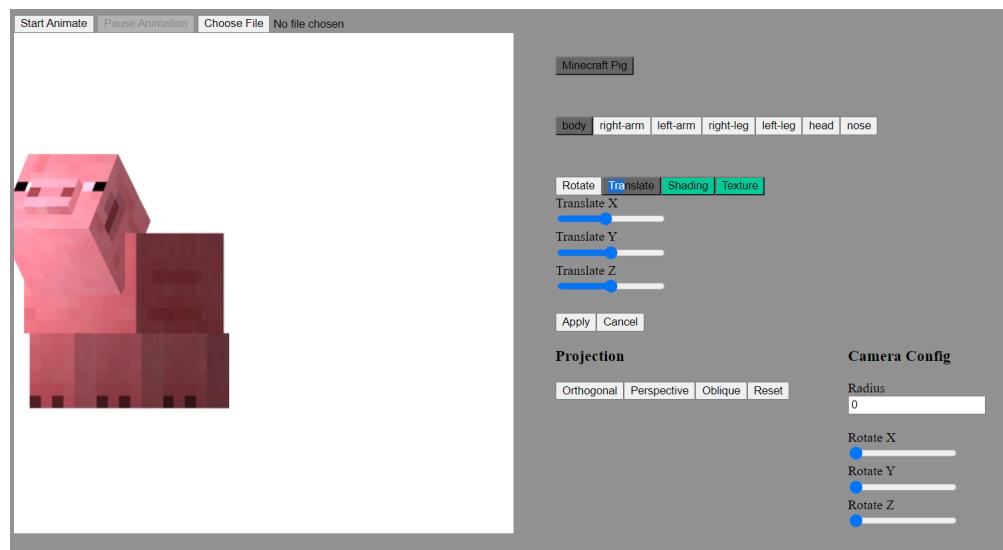


- a. Rotasi



Rotasi di atas dilakukan pada model Pig pada bagian “head” pada sendi rotate X. Karena rotasi dinaikkan, muka dari bagian “head” menjadi mengarah ke atas. Selain itu, bagian “nose” juga ikut mengalami rotasi karena merupakan *child* dari bagian “head”.

#### b. Translasi

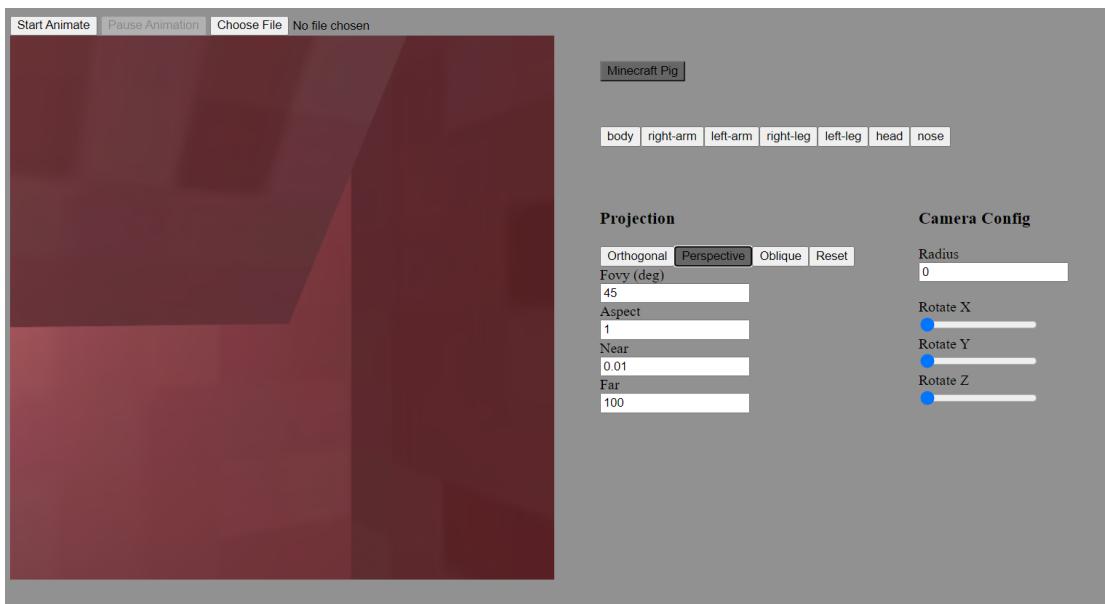


Translasi di atas dilakukan pada model Pig pada bagian “body” pada sendi translate X. Karena translasi pada sumbu X diturunkan, seluruh bagian model menjadi berpindah ke kiri karena bagian “body” merupakan bagian utama.

### 3. Mengubah jarak radius kamera dan berputar

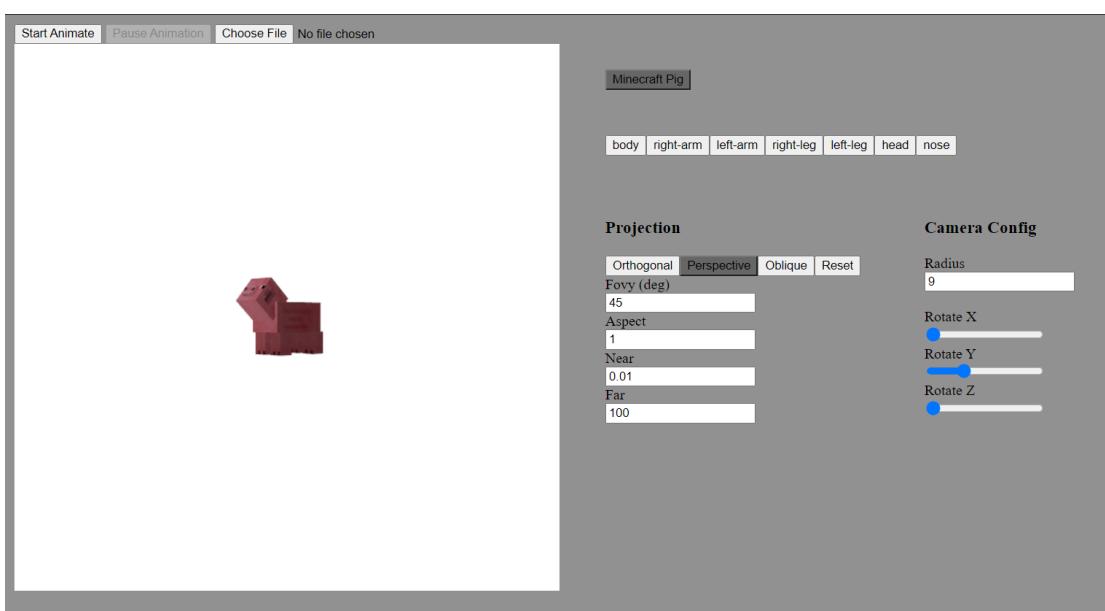
Model Pig dengan proyeksi *perspective* dengan parameter *default* dapat dilihat pada gambar di bawah.

Gambar awal:

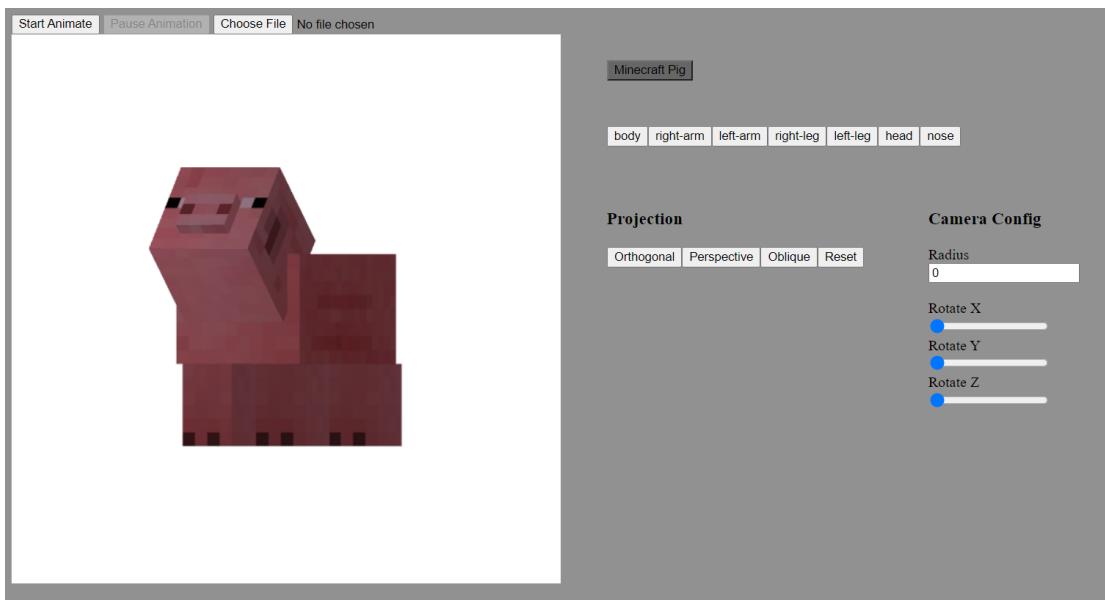


Kamera yang mengamati model tersebut diubah konfigurasinya dengan *radius* bernilai 9 dan *rotate* Y sekitar 120 degree. Dapat dilihat pada gambar di bawah, kamera seperti di-zoom out sehingga model Pig tampak kecil.

Gambar setelah diatur:



#### 4. Me-reset ke default view



Model Pig di atas dapat di-reset ke proyeksi semula dengan menekan tombol “Reset”.

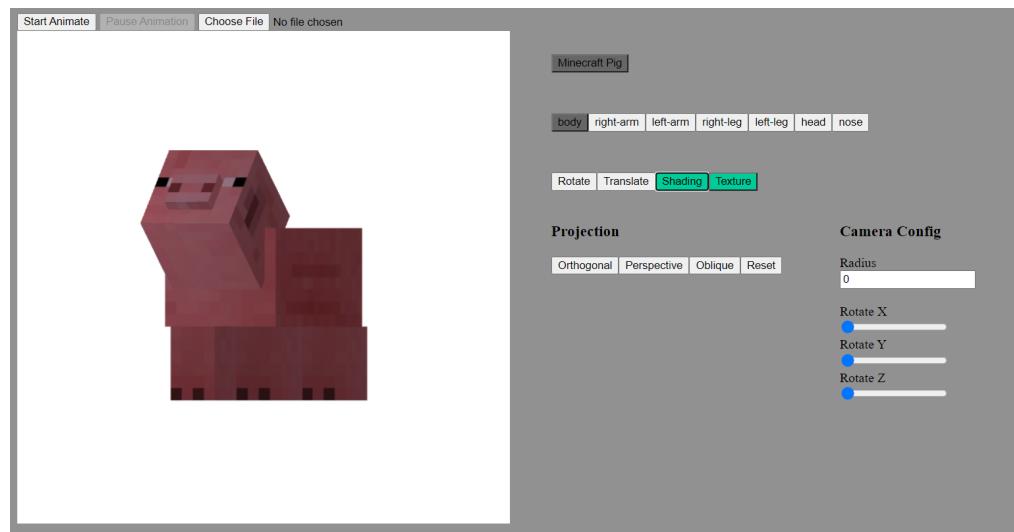
##### 5. Turn On/Off shading & texture

Untuk mengimplementasikan *turn on/off shading* dan *texture mapping*, kami mendefinisikan 2 boolean yang menandakan bahwa *shading* dan *texture mapping* sedang aktif pada shader. Boolean ini akan diubah nilainya saat tombol ditekan.

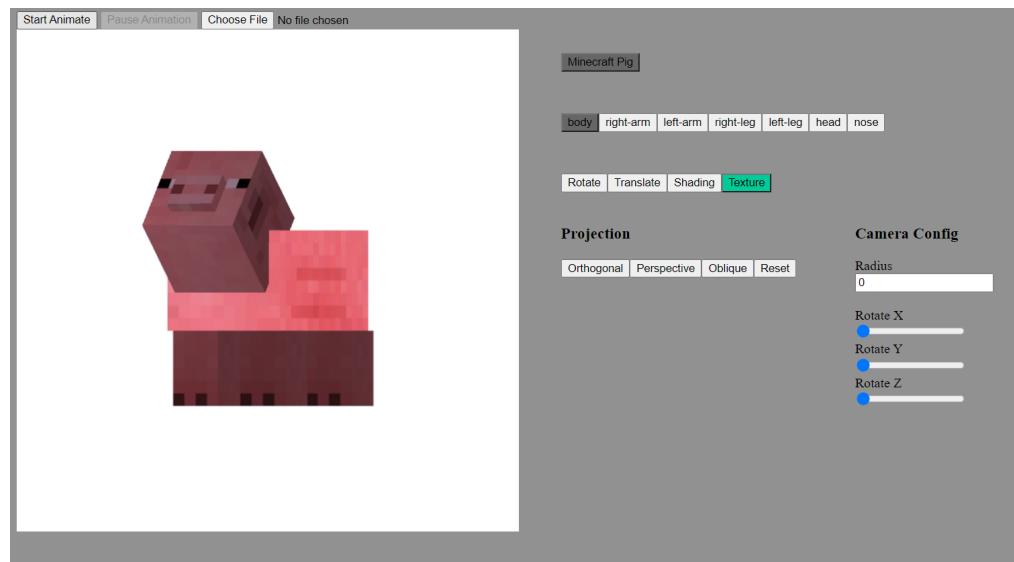
Berikut adalah tampilan *on/off shading & texture* pada setiap model:

###### a. Pig - Image Mapping

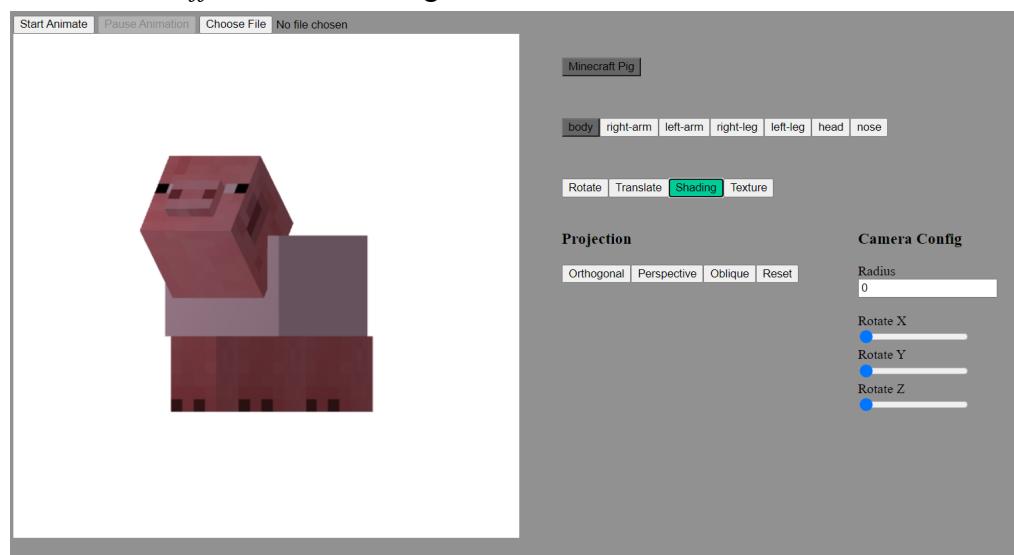
Gambar *turn on shading & texture* untuk Pig:



Gambar *turn off shading* untuk Pig:



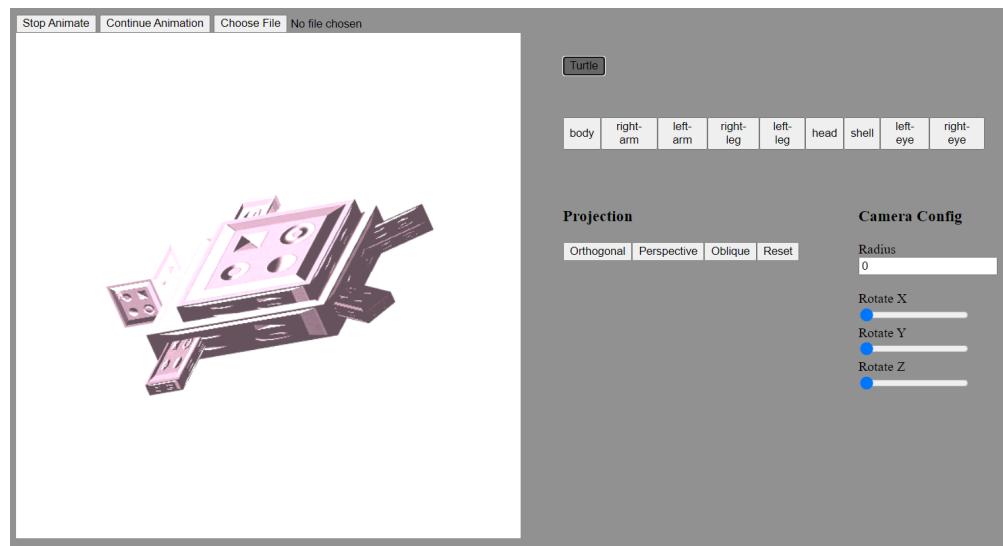
Gambar *turn off texture* untuk Pig:



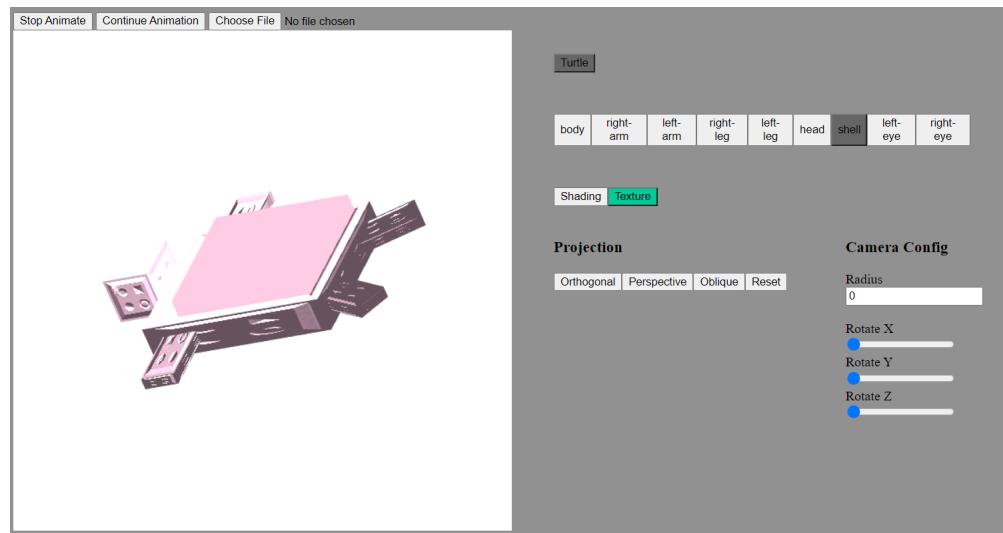
Dapat dilihat bahwa tekstur gambar pada model Pig tidak terlihat pada bagian "body".

- b. Turtle - Bump Mapping

Gambar *turn on shading & texture* untuk Turtle:

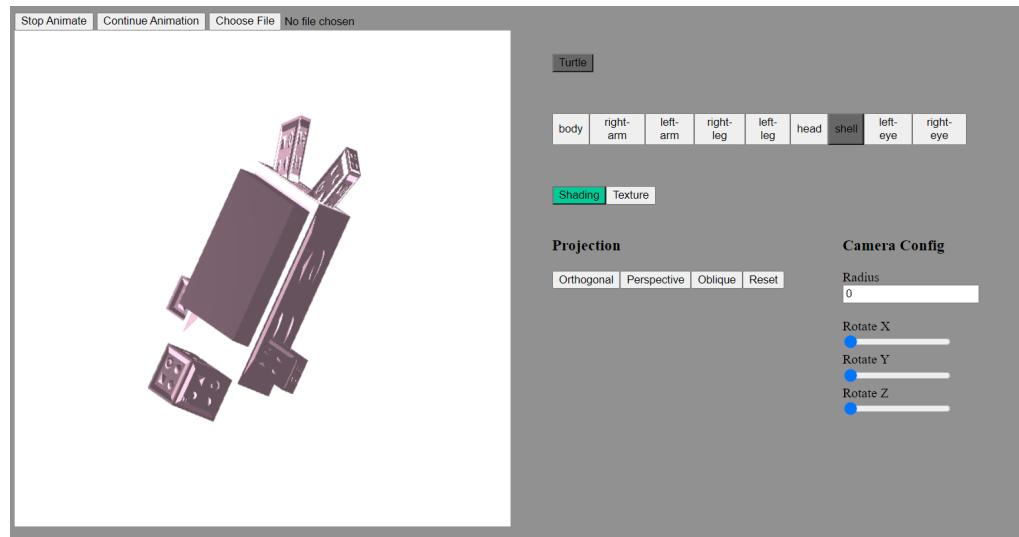


Gambar *turn off shading* untuk Turtle:



Dapat dilihat pada bagian “shell”, *bump mapping* tidak dapat terlihat karena shading tidak menyala sehingga tidak ada *lightning* yang membedakan vektor normal akibat pertubasi (bump).

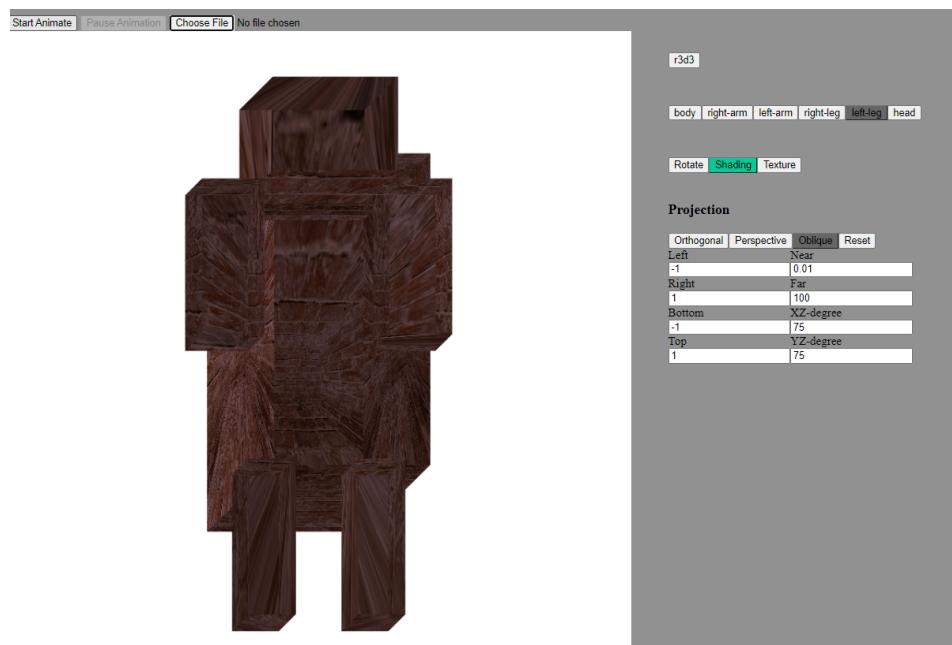
Gambar *turn off texture* untuk Turtle:



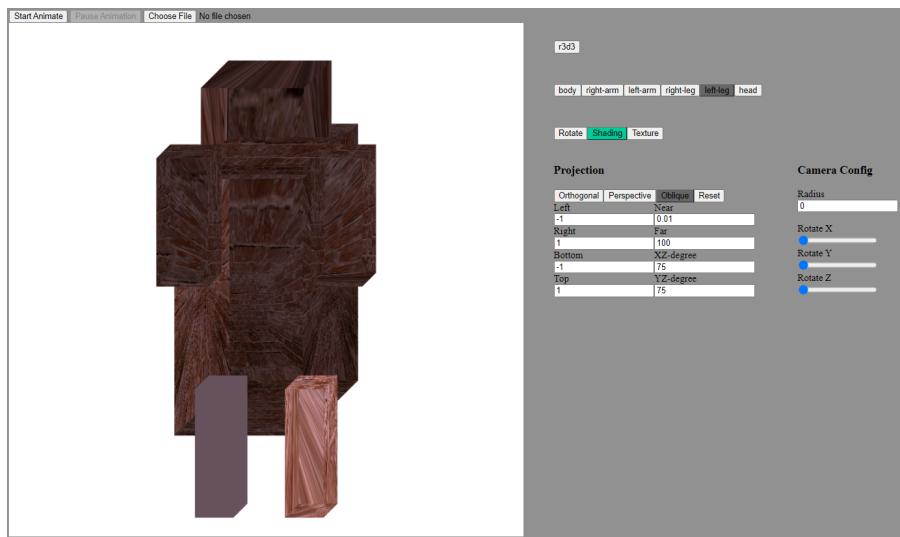
Jika dibandingkan dengan gambar awal, dapat dilihat perbedaan *shading* pada sisi yang sama (pada bagian “shell”) pada saat *bump mapping* diaktifkan.

### c. R3D3 - Environment Mapping

Berikut tampilan model dengan environment mapping dan kondisi di mana texture dan shading diaktifkan kedua-keduanya.



Bandingkan dengan tampilan di bawah di mana kondisi texture mode *diturn off* untuk left-leg, dan shading *diturn off* untuk right-leg



## 6. Turn On/Off animation

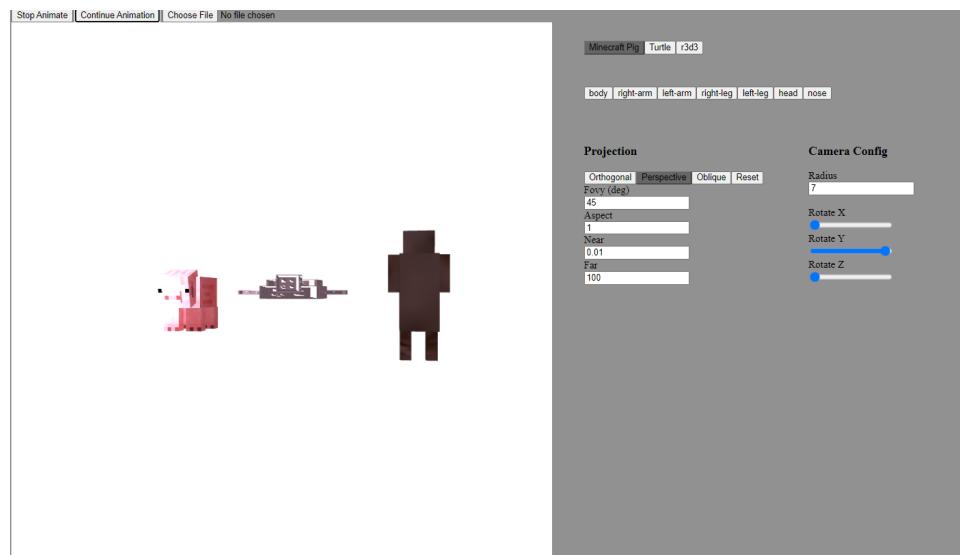
Animasi kami diimplementasi dengan melakukan kalkulasi selisih *state* antara 2 *frame* yang mengapit *frame number* untuk iterasi saat ini, misalnya *frame number* = 3 dan terdapat 3 *frames* dengan *frame\_number* = [1,5,10], kita hanya akan meninjau *frame* 1 dan 5 saja. Dari selisih state kedua frame ini, kita akan mengambil *progress* sebesar  $\frac{\text{frame\_num\_now} - \text{before\_frame\_num}}{\text{after\_frame\_num} - \text{before\_frame\_num}}$  yang dalam kasus di atas bernilai sebesar  $\frac{3-1}{5-1} = 0.5$ , yaitu 50% menuju frame 5. Kemudian hitung state objek tersebut memanfaatkan state sekarang dan hasil perkalian delta dengan *progress*.

Pada saat animasi sedang berjalan atau berhenti, *world view* secara otomatis diatur ke dalam proyeksi *perspective* dengan parameter default dan konfigurasi kamera dengan *radius* bernilai 7 dan *rotate Y* sekitar 350 degree.

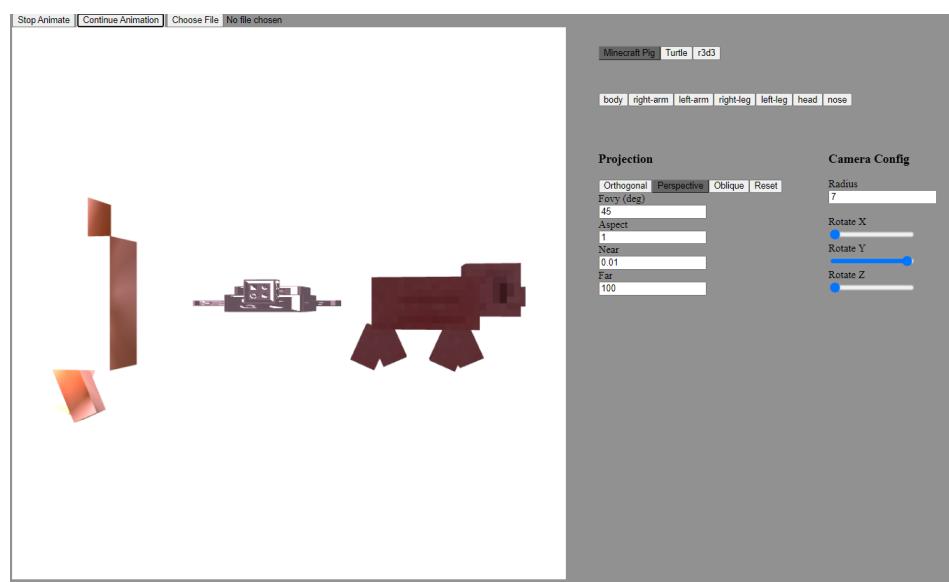
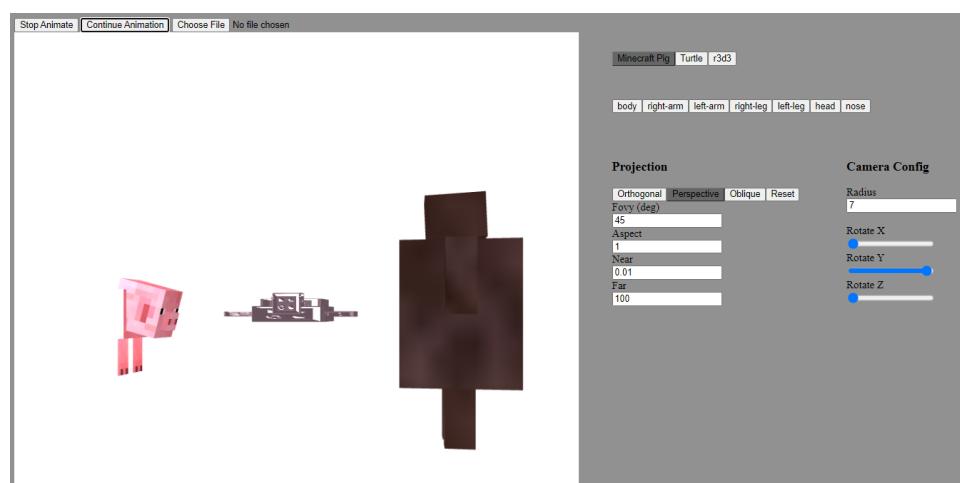
Animasi pada model Pig dilakukan dengan berjalan berputar, sedangkan pada model Turtle dilakukan dengan berjalan maju-mundur, sedangkan untuk animasi pada model R3D3 menggunakan animasi mengelilingi ruangan secara beraturan dan diakhiri dengan berputar 2 putaran sambil berjalan kembali ke titik awal.

Berikut adalah contoh animasi yang dilakukan pada model Pig dan Turtle:

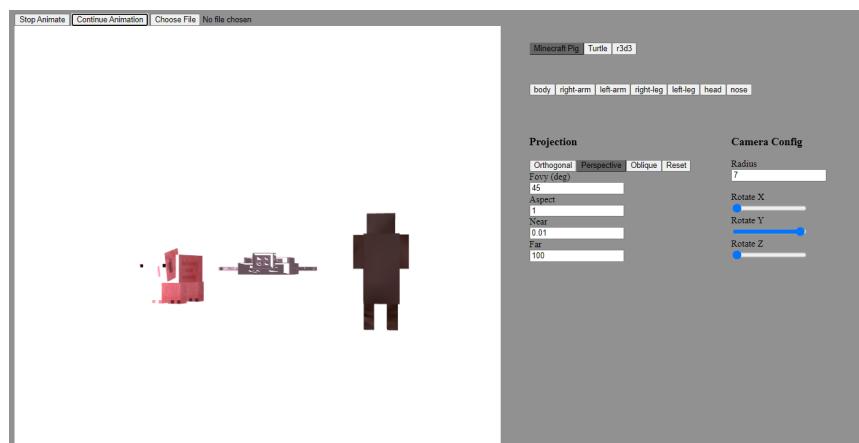
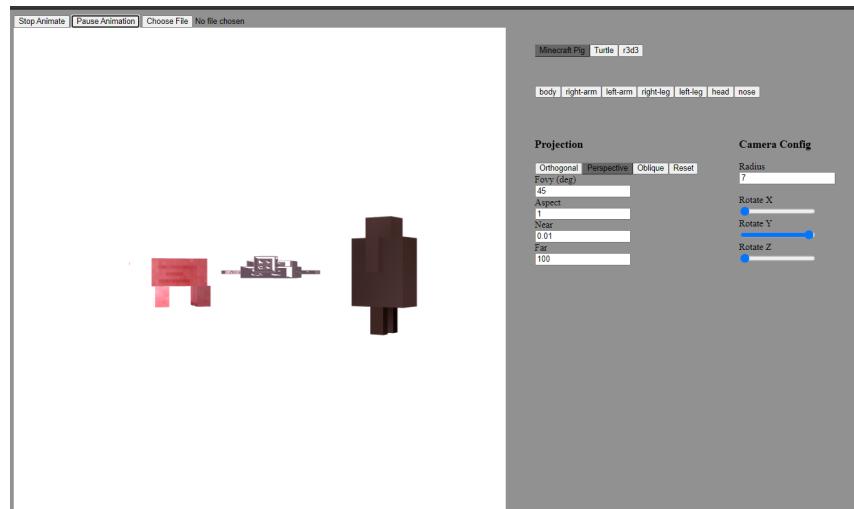
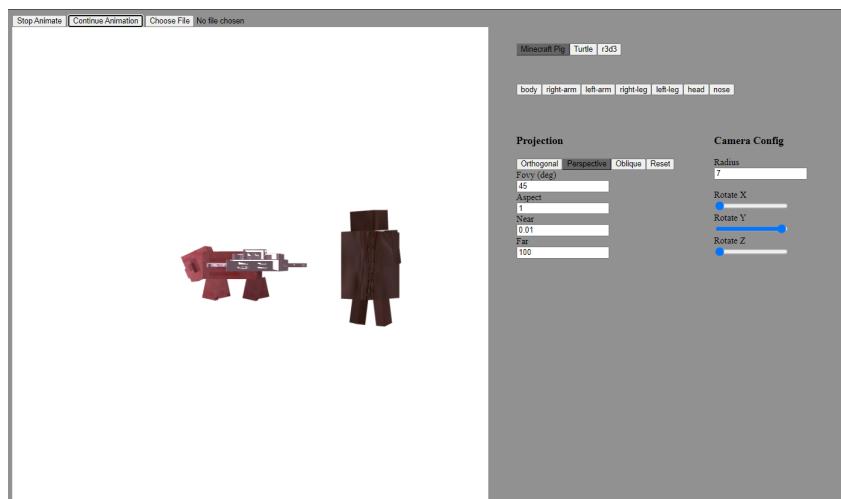
- Pada saat animasi dijalankan pertama kali (*paused*)



b. Pada saat animasi sedang berjalan (*paused*)

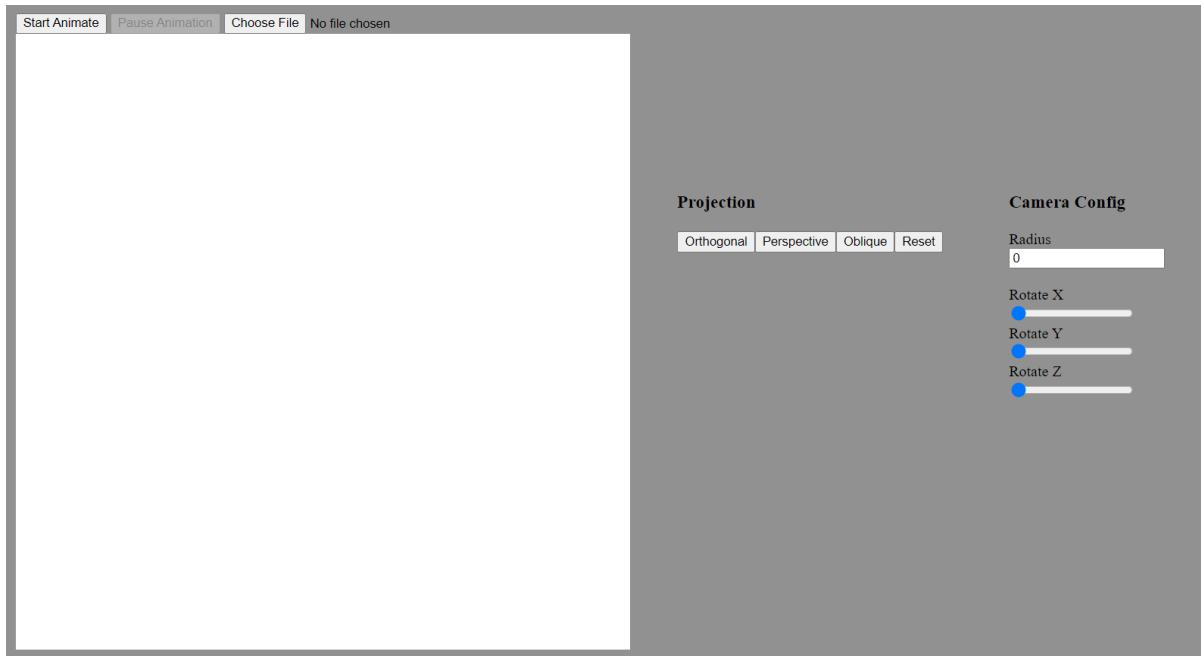


c. Pada saat animasi telah selesai, dilakukan looping



### BAB III

## CONTOH FUNGSIONALITAS PROGRAM



Berikut merupakan manual dari fungsionalitas program yang kami buat:

- Load model
  - 1. Pada ujung kiri atas, kliklah tombol load;
  - 2. Pilihlah berkas JSON model yang bersesuaian, lalu klik load;
  - 3. *Articulated model* akan muncul di kanvas.
- Manipulasi bagian-bagian model
  - 1. Setiap Anda meload model, akan muncul tombol di sebelah kanan kanvas untuk setiap model ke-i;
  - 2. Untuk memilih model ke-i, cukup tekan tombol model ke-i (misalkan Pig) dan akan muncul tombol baru berupa nama-nama bagian model.
  - 3. Untuk memilih suatu bagian model, cukup tekan tombol yang berisi nama bagian model dan akan muncul mode manipulasi sesuai sendi-sendii yang tersedia (translasi, rotasi);
  - 4. Untuk setiap mode, akan disediakan parameter-parameter tersendiri untuk mengatur manipulasi yang Anda inginkan;
  - 5. Jika Anda sudah puas dengan manipulasinya, tekan tombol apply untuk membuat manipulasi Anda permanen. Apabila Anda tidak jadi memanipulasi bagian model, Anda dapat menekan tombol cancel.
- Rotate viewer camera
  - 1. Anda dapat “menggerakkan kepala” Anda sebagai viewer dengan menggerakkan mouse Anda di kanvas sambil menekan mouse Anda untuk melihat sekeliling Anda di posisi Anda sekarang.
- Pengubahan jenis proyeksi

1. Di bawah opsi manipulasi objek, terdapat opsi untuk mengubah jenis proyeksi: orthogonal, perspektif, atau oblique;
  2. Klik jenis proyeksi yang Anda inginkan;
  3. Setiap jenis proyeksi memiliki parameter yang bisa Anda atur sesuai keinginan. Proyeksi akan otomatis berubah sesuai dengan parameter yang Anda atur;
  4. Untuk mengembalikan proyeksi ke view semula, klik tombol reset.
- Pengubahan radius kamera dan rotasi kamera
    1. Di bawah opsi manipulasi objek (dan di samping opsi pengubahan proyeksi), terdapat opsi untuk mengubah radius dan rotasi posisi;
    2. Terdapat 4 parameter: radius kamera terhadap titik origin, dan rotasi posisi kamera di tiga sumbu x, y, dan z terhadap titik origin. Aturlah parameter ini sesuai dengan keinginan Anda;
    3. Untuk mengembalikan keadaan kamera ke kondisi awal, yaitu radius nol (posisi di origin) menghadap ke sumbu-z negatif, klik tombol reset di bagian proyeksi.
  - Turn on/off shader
    1. Setiap Anda meload objek, akan muncul tombol di sebelah kanan kanvas untuk setiap objek ke-i;
    2. Untuk memilih objek ke-i, cukup tekan tombol objek ke-i dan akan muncul tombol baru berupa “shading” yang berwarna hijau;
    3. Tekan tombol ini untuk mengubah shading menjadi on/off;
    4. Warna hijau menandakan bahwa shading on dan warna putih menandakan bahwa shading off.
  - Turn on/off texture
    1. Setiap Anda meload objek, akan muncul tombol di sebelah kanan kanvas untuk setiap objek ke-i;
    2. Untuk memilih objek ke-i, cukup tekan tombol objek ke-i dan akan muncul tombol baru berupa “texture” yang berwarna hijau;
    3. Tekan tombol ini untuk mengubah shading menjadi on/off;
    4. Warna hijau menandakan bahwa texture on dan warna putih menandakan bahwa texture off.
  - Turn on/off animation
    1. Pada saat animasi sedang mati, Anda dapat menjalankan animasi dengan menekan tombol “Start Animate” pada kiri atas.
    2. Pada saat animasi sedang berjalan, Anda dapat menekan tombol “Stop Animate” untuk mematikan animasi atau “Pause Animation” untuk menghentikan animasi.
    3. Animasi yang sedang berhenti dapat dilanjutkan kembali dengan menekan “Continue Animation”.

## **BAB IV**

### **PEMBAGIAN KERJA**

Berikut merupakan pembagian kerja yang dilakukan dalam pengeroaan tugas:

<b>Nama</b>	<b>NIM</b>	<b>Tugas</b>
Felix Setiawan	13518078	<ol style="list-style-type: none"><li>1. Membuat model Turtle</li><li>2. Membuat laporan</li></ol>
Yonatan Viody	13518120	<ol style="list-style-type: none"><li>1. Membuat implementasi articulated model</li><li>2. Membuat load &amp; parse model</li><li>3. Membuat transformasi model</li><li>4. Membuat implementasi animasi</li><li>5. Membuat implementasi image mapping</li><li>6. Membuat implementasi bump mapping</li><li>7. Membuat implementasi environment mapping</li><li>8. Membuat implementasi shading</li><li>9. Membuat model view kamera</li><li>10. Membuat model Pig</li><li>11. Melakukan debugging</li><li>12. Membuat laporan</li></ol>
Stephen Thajeb	13518150	<ol style="list-style-type: none"><li>1. Membuat model R3D3</li><li>2. Membuat laporan</li></ol>

## **REFERENSI**

Spesifikasi Tugas Besar 3 Grafika Komputer IF3260.

[https://drive.google.com/file/d/1Zqdv-84nnv54MkU7hmtPx6q8FNi7\\_wsb/view?usp=sharing](https://drive.google.com/file/d/1Zqdv-84nnv54MkU7hmtPx6q8FNi7_wsb/view?usp=sharing).

Diakses 1 April 2021, 21.05

Angel, Edward dan Dave Shreiner. *Interactive Computer Graphics: A Top-Down Approach with WebGL*. 2015. London: Pearson Education.