

Block Fabrik

Boris Asenov (01500627), Caroline Magg (01225388)

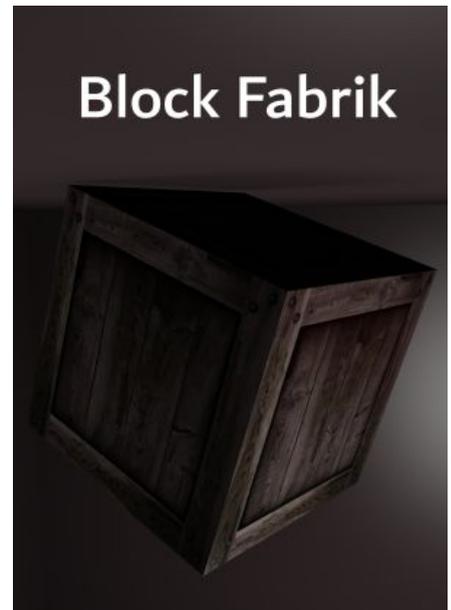


Table of content

- Story & Walkthrough
 - Features
 - Effects
 - Objects
 - Key and mouse handles
 - Cheat shortcuts for Debugging
 - Code structure overview
-

Story

You, a cube awakened from its inanimate slumber, have set out to find knowledge of the outside world. On your journey you will have to use the help of your brethren to escape the Block Fabrik.

Walkthrough

Your goal is to escape the Block Fabrik. For testing purposes you can use the number keys (1, 2, 3, 4, 9, 0) to teleport to different locations and skip through levels. All directions used in this walkthrough like “go left”, “go up” are relative to the entry point of the level.

Part 1 (press 1 to teleport to the start, press 2 to skip the level)

Your goal is to exit through the opening on the upper floor. (A spotlight is pointed to the exit in each level)

1. Use WASD and the mouse to hop down from the conveyor belt. (Look to the right and press ‘W’ to move in that direction)
2. Push over the 4 boxes (if they aren’t already) so that you can climb on them. Use the SPACE key to get a little boost which enables you to push and climb (Upon pressing space you boost for a short time ~0.25 sec and then have to wait out the cooldown (0.75 sec) to be able to boost again). It takes a bit to get used to being a cube, don’t give up!
3. Use SPACE and the WASD keys to climb on one of the boxes and try to hop on to the moving elevator.
4. Hop off the elevator onto the upper platform and go through the hole in the wall

Part 2 (press 3 to skip this part, press 2 to start again at the entrance)

Your goal is to exit the room via themaschine on the left side of the room (a spotlight is pointed at it)

1. Hop down from the upper platform to the ground floor.
2. Get on the conveyor belt in front of you. (Use the momentum of the conveyor belt and the SPACE button to help yourself get up there)

3. Stay on the conveyor belt and go through the rusty gray machine (through the gray opening). You should get teleported to the next level.

Part 3 (press 4 to skip this part, press 3 to start again at the entrance)

Your goal is to exit the level through the door on the upper floor. Notice that your cube changed, it now has a metal frame!

1. Hop off the conveyor belt (or at least don't let it push you to the other end, if it does, you lose the game and have to start again).
2. Notice how some boxes drop down from the dispenser (there can be no more than 4 boxes in the level).
3. Push at least one of the boxes to the left side of the conveyor belt.
4. Push the box close to the big cylinder on the wall. Use SPACE to make it easier.
5. Climb on the box.
6. Because of your new metal sides you have magnetic properties and can stick to metal by holding LEFT-SHIFT.
7. Jump from the box and stick to the cylinder using shift. It takes some practice.
8. Climb on the cylinder and jump to the grey metal wall on its side (don't forget to keep holding the SHIFT button)
9. Climb on the upper platform, exit through the hole in the wall

Part 4 (press 9 to skip this part and go straight to the end of the game, press 4 to start again at the entrance)

Your goal is to get on the upper platform on the left side of the room (relative to the platform you are standing on).

To do that you have to use what you have learned in the last room and climb some metal! (using SHIFT of course)

1. Hop straight down to the floor and continue straight to the gate on the other side of the room
2. Use SHIFT to climb on said gate
3. Climb to the upper left corner and transition over to the ventilation vents. (keep holding that SHIFT button)
4. Perform a balancing act and follow the path of the ventilation vents.
5. Transition from the vents to the gray metal wall (similar to the one from the previous level).
6. Transition from the metal wall to the wooden platform and go through the hole in the wall.
7. Enjoy some fireworks!

Features

Camera

The first camera mode is a fixed mode used for the game itself. The camera has the character as focus point and is rotating around it, when the mouse is moved. Scrolling the mouse wheel causes a zoom in or zoom out on the object in focus. With F3 the camera mode can be changed. The free mode allows the camera to move unrestricted in the scene. This mode can be used for debugging purpose to inspect the scene.

Character cube

The main character for the first levels is a moving cube with evolving properties. The properties such as the material and related skills are changing as the levels are solved.

The movement is controlled by the gamer and realized by the following steps:

- reading the keyboard inputs
- add according to the pressed keys force or torque
- calculate the physical movement of the character (ie updating the physical scene)
- fetch results of physical scene and update the model matrix accordingly
- render the object with the current model matrix

The character cube has its own shader, which uses specular maps (for implementation details see 'Effects'). In the first two levels, the cube has a wood-only texture and the specular map is just black to avoid strong reflection. After entering level 3, the cube's texture changes to wood with reflecting metal frame.

PhysX

As mentioned, the PVD debugging tool can be used for our application. The tool needs to be started before the source code is started.

Window

The screen resolution of the normal screen mode and the full screen mode, the refresh rate, the screen mode (full screen or not), the brightness and the title can be changed by changing the settings file and without recompiling the code. Toggling the full screen mode is possible with F5. The window will be positioned at the left upper corner of the screen. The default for the brightness coefficient is set to 1.0. Increasing the multiplier makes the scene brighter, decreasing the value will result in a darker scene.

Model

The model uses the ASSIMP library to read in .obj (and connected .mtl) files. [14] The gathered geometry, lighting and texture information is used for the purpose of creating an Geometry object which can be rendered (see Geometry). Additionally if the "Texture" folder contains another .obj file which has the same name but with the "CH" prefix added (e.g. "spaceship.obj" & "CHspaceship.obj") that file is used to load a dynamic convex hull physx object which acts like a "hitbox" and allows for simulation (the Geometry objects moves with its hitbox) (is not used in game).

It is also possible to load in a triangle mesh via .obj file (used for the cylinder in level 2).

As last addition the Model class also supports importing .obj files with box information and creating collision boxes from it. The whole map (in terms of collision) has been imported as single .obj file. [15]

Conveyor Belts

For the 2 conveyor belts in the game we have set up long kinematic boxes which move at a constant speed in a set direction. For each conveyor belt, 2 of these boxes are used so that at least one of them is in the part with which the player can interact at all times. They create an infinite loop of teleporting behind each other so that the conveyor belt in the level never stops moving.

Dispenser

A "box dispensing system" for Level 3 of the game has been implemented. It ensures that the room will be filled with all of the implemented boxes (in this case 4) by checking every few seconds if any boxes are not in the room yet and teleporting one of them if that is the case. Noteable is that the amount of cubes can be increased with minimal effort.

Shaders

The basic phong-based shader is used for some objects in level 1 and the models of the levels.

Furthermore, there are two shaders implementing PBR with color or texture used by several objects in all levels and a shader for specular maps.

All shaders support more than one light source (direction light and spotlight). In the current code, the maximal number of direction light sources is set to 1 and of spotlights to 2. The shader is extended to take the shadow maps into account (more details on implementation in 'Effects').

Lights

There are different light sources in different levels:

Directional light	The flashlight is the same for all levels. The direction is (0, -1, -1) with color (0.3, 0.3, 0.3).
Flashlight	The flashlight is in the position of the camera and points in the front direction. It follows the cube through all levels and can be toggled with the key F.
Spotlight	Each level has its own spotlight location to provide a hint where to go next. Part 1: The spotlight is located above the exit to level 2 with the position (5.0, 10.5, -11.5) and the direction (0.0, -1.0, -1.0)

	<p>Part 2: First entrance from level 1, the spotlight is pointing towards the transition box to level 3. The position is (-15.0, 7.5, -36.0) with direction (-1.0, -1.0, 0.0).</p> <p>Part 3: The spotlight is located at (-33.0, 10.5, -23.5) with direction (1.0, -1.0, 0.0), which results in illuminating the exit to level 2.</p> <p>Part 4: The spotlight changed its location to (-15.3, 11.5, -40.5) with direction (0.0, -1.0, -1.0) to illuminate the exit.</p> <p>Win platform: The spotlight has location (199.0, 6.5, 4.5) and direction (0, -1, -0.5).</p> <p>Loose platform: No spotlight</p>
--	--

The inner cutoff angle for both spotlights is 15.5 and the outer cutoff angle is 20.5. Between these two angles the intensity is linearly interpolated to dampen the lightening going outwards in the light cone. The attenuation coefficient for both spotlights are 1.0, 0.04, 0.01.

Effects

View Frustum Culling

The view frustum culling functionality is based on [1]. This approach uses enclosing spheres of objects which are compared with the 6 planes of the view frustum. For our game, we used the following realization. The enclosing spheres are generated at object creation by taking the mean of all object vertices as sphere center and the maximal distance between object's vertices and sphere center as sphere radius. The center and radius are first generated in model space. At the time of the actual comparison, the data is queried from the object and multiplied with the current model matrix, transforming the enclosing sphere to world coordinates. The view frustum planes are defined by a point on the plane and the normal vector. Thus, the distance between a point and a plane can be calculated by $N \cdot p + (-N \cdot P)$, where N is the normal vector of the plane, P is a point on the plane, p is the given point to compare. To take the radius of the enclosing sphere into account, the culling logic looks like this:

```
Intersect  If one distance < radius
Outside   If one distance < - radius
Inside    Else
```

For models with more than one geometrical mesh, this logic is extended:

```
Intersect  If one of the meshes intersects
Outside   If no mesh is either inside or intersected
Inside    If all meshes are inside
```

The character cube is not considered when performing view frustum culling, since the game mode of the camera is fixed on the cube and therefore the cube is always drawn. The same applies to the level models, they are only rendered if the cube is in the corresponding level. According to the global position of the cube, the rendering of the level model and the objects inside the room is switched to avoid unnecessary rendering or view frustum comparisons.

Shadow maps with PCF

For the implementation of the shadow maps, we mainly used [2], [3] and the provided repertorium slides. Shadow maps are generated for the directional light and for the spotlight, which is at a fixed location for each level. Calculating shadows for the flashlight would not make sense, since the casted shadow is never visible from camera position but would need additional render passes and therefore time. The border value for both textures is set to 1.0 to avoid shadows outside the depth map's range. The shadow artefacts are compensated with PCF with linear filtering (integrated OpenGL functionality)

and culling front-facing triangles for all shaders and depth maps. In addition to that, a small bias for the depth comparison is used to avoid shadow acne effects when the cube is moving.

The shadows of directional light can be seen in all levels. The shadow of a spotlight can be seen for example in level 2 after the first entrance at the transition box to level 2. The light specific values like position and direction for view frustum of depth render pass are either taken from the light specification (for spotlight, see 'Features – Light') or are adapted per level (direction light). To reduce the render calls, it is tried to only render objects to the depth maps which cast a shadow.

CPU Particle Systems

[4], [5] and [6] were used to implement a CPU particle system. Different to the tutorial [4], the three buffers for mesh vertices (stays the same for all particles), position and color are bind to one VAO, which is then used for instancing. Each frame a predefined number of particles is awaken at a fixed starting position and with the color red. The velocity with the main direction is set randomly. Over the lifetime of a particle the position is updated according to the velocity which is decreasing each frame taking gravity into account. Additionally, the color is linear interpolated from red to green to blue over the lifetime.

Specular map

Specular map was implemented with the help of [7]. This is also the source for the used character cube texture. The specular map is used for the specular light part of both the directional light and the spotlights. Therefore, the effect can be seen best, if the cube has its metal-framed wood texture and the camera flashlight is pointed at the metal frame.

Physically based shading

Physically based shading or physically based rendering (PBR) was implemented based on [8], [9] and [10]. The PBR for our game is based on the 4 parameters of [9]: albedo, ambient occlusion, roughness and metallic. There are two versions of the PBR shaders, depending on the type of the parameter albedo. Albedo can either be provided by an RGB color or a given texture. Additional to directional light, also spotlight illumination and shadows of both supported light sources are taken into account in the PBR shaders.

Objects

This part should cover all the rendered objects in the scene and provide information about their illumination, textures and loading (geometry and/or model type). As mentioned before, all rendered objects are illuminated by up to three light sources using the different shader programs.

Character – Cube

Object	Geometric mesh of a cube with physX object, weight = 10
Shader program	Specular map with directional light, spotlights and shadow maps (bias)
Texture	Before Level 3: wood texture and black specular map (adapted from [7]) After Level 3: wood texture with metal frame and specular map to highlight metal frame [7]

Level 1

Object	Blender model imported as obj file with textures in mtl file.
Shader program	Basic phong shading with directional light, spotlights and shadow maps
Texture	Floor texture, Wall texture, Top texture, Oven (1,2,3) Texture[13]

Stack of carton cubes

Object	Geometric mesh for 4 cubes with physX object, weight = 3
Shader program	PBR with texture and directional light, spotlights and shadow maps
Texture	Carton texture [11], roughness = 0.8, metallic = 0.01

Elevator Level 1

Object	Geometric mesh for cube with physX object
Shader program	Basic phong shading with directional light, spotlights and shadow maps
Texture	Floor texture [13]

Metal cube with texture Level 1

Object	Geometric mesh for cube with physX object, weight = 100
Shader program	PBR with texture and directional light, spotlights and shadow maps
Texture	Metal texture [12], roughness 0.1, metallic 1.0

Wood cube Level 1

Object	Geometric mesh for 4 cubes with physX object, weight = 10
Shader program	PBR with texture and directional light, spotlights and shadow maps
Texture	Wood texture with metal frame [7], roughness 1.0, metallic 0.2

Metal cube without texture Level 1

Object	Geometric mesh for 4 cubes with physX object, weight = 100
Shader program	PBR with color and directional light, spotlights and shadow maps
Texture	No texture, color (2.0, 2.0, 2.0), roughness 0.1, metallic 1.0

Level 2

Object	Blender model imported as obj file with textures in mtl file.
Shader program	Basic phong shading with directional light, spotlights and shadow maps
Texture	Floor texture, Wall texture, Top texture, Gate texture[13]

Metal wall Level 2

Object	Blender model imported as obj file.
Shader program	PBR with color and directional light, spotlights and shadow maps
Texture	No texture, color (0.6, 0.6, 0.6), roughness 0.1, metallic 1.0

Metal shaft Level 2

Object	Blender model imported as obj file.
Shader program	PBR with color and directional light, spotlights and shadow maps
Texture	No texture, color (0.6, 0.6, 0.6), roughness 0.1, metallic 0.9

Machines Level 2

Object	Blender model imported as obj file with textures in mtl file.
Shader program	PBR with texture and directional light, spotlights and shadow maps
Texture	Metal10 texture, roughness 0.5, metallic 0.5

Transition Box Level 2

Object	Blender model imported as obj file with textures in mtl file.
Shader program	PBR with texture and directional light, spotlights and shadow maps
Texture	Metal20 texture, roughness 0.5, metallic 0.6

Level 3

Object	Blender model imported as obj file with textures in mtl file.
Shader program	Basic phong shading with directional light, spotlights and shadow maps
Texture	Floor texture, Wall texture, Top texture, Metal43 Texture[13]

Metal wall Level 3

Object	Blender model imported as obj file.
Shader program	PBR with color and directional light, spotlights and shadow maps
Texture	No texture, color (0.9, 0.9, 0.9), roughness 0.1, metallic 1.0

Metal cylinder Level 3

Object	Blender model imported as obj.
Shader program	PBR with color and directional light, spotlights and shadow maps
Texture	No texture, color (0.9, 0.9, 0.9), roughness 0.1, metallic 1.0

Transition Box Entry and Exit Level 3

Object	Blender model imported as obj file with textures in mtl file.
Shader program	PBR with texture and directional light, spotlights and shadow maps
Texture	Metal20 texture [13], roughness 0.5, metallic 0.6

Falling carton cubes Level 3

Object	Geometric mesh for 4 cubes with physX object, weight = [1, 3, 3, 15]
Shader program	Basic phong shading with directional light, spotlights and shadow maps
Texture	Carton texture[11] , Wood texture [7], Metal texture [12]

Win and lose condition

The win condition is to exit the room of level 2 through the upper opening.

The player loses the game if he navigates the cube through one of the wrong exits. In level 1, the game is lost if the player does nothing and the cube is transported through the lower opening into the oven via the conveyor belt. In level 2 there is no explicit losing condition, the room is designed in a way, that there are only two possible ways out of the room, both leading somewhere save. The lose condition in level 3 is similar to level 1 doing nothing and let the cube be transported on the conveyor belt to the exit transition box.

Key and mouse handles

Key	Function
F1	Toggle Wireframe
F2	Toggle backface culling
F3	Toggle camera mode
F5	Toggle fullscreen mode
F8	Toggle view frustum culling
F9	Toggle console output of view frustum culling
WASD	Character movement front, left, back, right
F	Toggle flashlight
Space	Gives the character a small boost for limited amount of time (after which a cooldown has to be waited out) , which allows for easier climbing and pushing
Shift [unlockable]	Gives the character additional force, which allows a metallic framed cube to climb metallic walls.
Up, down, left, right arrows	Camera movement in free mode

*Some key handles are only used for debugging purpose (eg. F1, F2, F3, F9).

Mouse	Function
Moving the mouse	In fixed mode: rotate around the object In free mode: moves pitch/yaw of camera
Scrolling the mouse wheel	In fixed mode: Zoom in and out with character in focus In free mode: -

Note: The functionality depends on the current camera mode. The fixed mode is the game mode, the free mode is used for debugging.

Cheat shortcuts for Debugging

The following cheat shortcuts provide the opportunity to skip single levels. They are realized by teleporting the character cube to a specific location in each level.

Key	Function
1	Teleport to start of level 1
2	Teleport to first start of level 2 coming from level 1
3	Teleport to start of level 3. After this shortcut the character cube has a metal-framed wood texture and its abilities change.
4	Teleport to second start of level 2 coming from level 3
9	Teleport to platform win
0	Teleport to platform loose

Code structure overview

The Source Code is structured in different classes, which are listed below in alphabetical order. The main basic structure is adapted from the provided ECG framework, but the classes are all implemented by us (partly based on additional tutorials).

Camera

There are two different camera modes, which are described in more detail in 'Features'. The camera class contains the methods for the camera movement, the camera direction calculations and setter/getter needed by other classes.

Character

A character object is the game character controlled by the gamer. It's a rigid dynamic PhysX object with mass 10. The imaginary, only for physical calculation used PhysX object is visualized by a rendering (geometry) object.

A character object has everything needed to calculate and render the character – the needed PhysX objects, a material property, a geometry object, its initial position and size.

Frustum

The frustum class implements everything necessary for view frustum culling (more details on implementation in 'Effects'). This includes the calculation of the view frustum, the comparison between points and enclosing spheres of objects to view frustum and wrapper methods for the draw call of objects and models.

Geometry

A geometry object contains everything needed to render a vertex object. This includes the mesh data information of the position of the vertices, the indices defining a face and the normal and uv coordinates for each vertex. Additionally, the model matrix for the overall position in space, a texture object, a vector defining material coefficient for light calculation and the used shader are provided. For a PhysX object is the dynamic rigid body actor given to fetch the current position in the physical scene. The class handles updating the model matrix for rendering (in sync with physical object), building the VAO and setting the correct attributes in the assigned shader (rendering the object).

Since Meshdata is only a struct to collect vertices, indices, normals and uv coordinates information, the geometry class provides methods to create basic structures such as cube, pyramide and plane.

Model

Model objects are used as addition to geometry objects. The main difference is, that geometry objects are provided by (manually) created triangle meshes and model objects are objects with mesh data loaded from files (more details about the loader in 'Features').

Particles

The particles class handles the creation, update and rendering of a particles system with a given starting position and a fixed update logic (more details on implementation in 'Effects').

PhysX

This class is used to externalize the physX initialization and the activation of PVD to a single class. The physX objects, which are required for other objects can be extracted by getter methods. This class should mainly serve an overview purpose.

Shader

The class provides methods to compile, add and link shaders, to use and clear them and to set uniforms of different types (currently used are: glm::mat4, glm::vec3, glm::vec4, float) at a name given location.

Texture

The texture class creates an object, which read a texture form a provided file, binds and activate it for rendering. A texture object must be provided to the geometry object. Textures have mipmapping and trilinear filtering enabled.

Window

This class is similar to PhysX used to externalize the OpenGL window initialization and should make the main file a bit less cluttered. The width and height are currently set to 1280x768 and are set in a settings file together with the window title, the refresh rate and if the window should be in fullscreen mode or not. The settings file is read in by INIReader, like in the ECG framework.

References

- [1] <http://www.lighthouse3d.com/tutorials/view-frustum-culling/>
- [2] <http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/>
- [3] <https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping>
- [4] <http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/>
- [5] <https://learnopengl.com/In-Practice/2D-Game/Particles>
- [6] <http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/billboards/>
- [7] <https://learnopengl.com/Lighting/Lighting-maps>
- [8] https://disney-animation.s3.amazonaws.com/library/s2012_pbs_disney_brdf_notes_v2.pdf
(Physically-Based Shading at Disney by Brent Burley, Walt Disney Animation Studios)
- [9] <https://cdn2.unrealengine.com/Resources/files/2013SiggraphPresentationsNotes-26915738.pdf>
(Real Shading in Unreal Engine 4 by Brian Karis, Epic Games)
- [10] <https://learnopengl.com/PBR/Theory> and <https://learnopengl.com/PBR/Lighting>
- [11] https://www.freepik.com/free-photo/cardboard-wallpaper-template_1037197.htm
- [12] https://www.freepik.com/free-photo/metal-plate-texture_4582396.htm
- [13] <https://www.cg.tuwien.ac.at/courses/Textures/>
- [14] <https://learnopengl.com/Model-Loading/Assimp>
- [15] <https://gameworksdocs.nvidia.com/PhysX/4.0/documentation/PhysXGuide/Manual/Index.html>