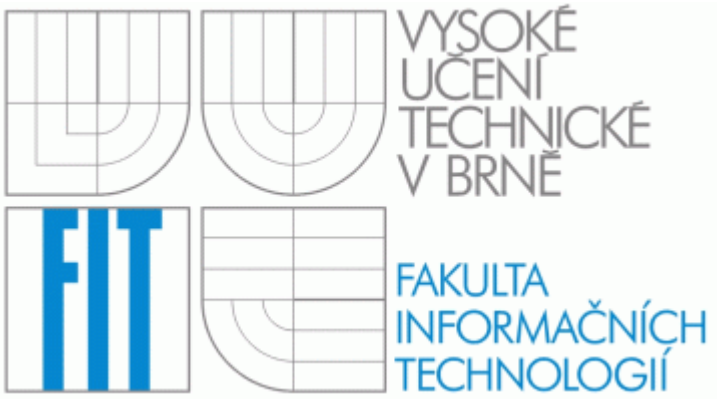


POČÍTAČOVÁ GRAFIKA - 1MIT

2007 / 2008

Projekt – Jednoklávesová hra s použitím OpenGL

ボールあつかいがうまい

Booruatsukaigaumai

(Jap) (adj) being good at handling a ball

Vydáno
21.11.2007

Autoři: Ivo Skalický, xskali03@stud.fit.vutbr.cz
Jaroslav Bartoň, xbarto42@stud.fit.vutbr.cz
Petr Dittrich, xdittr03@stud.fit.vutbr.cz
Jiří Tužil, xtuzil00@stud.fit.vutbr.cz

O projektu

Booruatsukaigaumai je realizací projektu do předmětu Počítačová grafika na FIT VUT Brno ve školním roce 2007/2008. Booruatsukaigaumai je výraz z japonštiny znamenající „být dobrý v ovládání míče“. Samotná hra je inspirována dětskou hračkou tvořenou destičkou s bludištěm upevněnou v průhledném plastovém krytu a kuličkou. Úkolem je dostat kuličku z jednoho místa bludiště do druhého. Naše hra je založena na podobném principu s tím, že celá destička je v jednom směru implicitně nakloněna. Naklonění do druhého směru je ovládána stisknutím nebo uvolněním klávesy mezerník. Protože hra má být ovládána pouze jednou klávesou, je destička na jednu stranu samovolně natáčena pokud není stisknuta herní klávesa. Po stisku herní klávesy se směr rotace mění. Na desce jsou umístěny blokové překážky ale i otvory skrz desku. Cílem hry je ovládat míčky dopadající na desku takovým způsobem, aby projely co nejvíc různě bodově ohodnocených branek. Projetí branky míčkem vyvolá navýšení skóre. Cílem je absolvovat všechny úrovně s co největším bodovým ziskem. Projíždění branek znesnadňují především otvory do desky, které mohou způsobit propadnutí míčku a jeho ztrátu. Zároveň je třeba dát si pozor na to, aby míček předčasně nespadl z desky. Jednotlivé úrovně jsou odstupňovány podle obtížnosti.

Podle zadání je celá hra realizována jedním binárním souborem. Veškeré úrovně tak musí být uloženy napevno ve zdrojovém kódu. Pro vytváření úrovní bylo potřeba vytvořit editor, který bude umožňovat vyexportovat vytvořenou úroveň do zdrojového kódu jazyka C. Zadání projektu také zakazuje použití cizích knihoven a nutnost přenositelnosti celého projektu mezi systémy Windows a Unix. To si žádá zvláštní přístup k vykreslování textového výstupu. Font použitý pro textový výstup tak musí být uložen ve zdrojovém kódu, z něhož bude později rekonstruován.

Zajímavou funkcí, kterou si požadavek jediného binárního souboru vyžaduje je také automatický konfigurátor kvality, který přímo za běhu programu snižuje detaily a kvalitu vykreslování podle dosažené hodnoty FPS[1].

Rozdělení práce v týmu

Ivo Skalický - xskali03

Vedoucí týmu, kostra projektu, vykreslení scény, generování stínů, vygenerování fontu pro přenositelné použití, automatický konfigurátor kvality, započítání skóre

Jaroslav Bartoň - xbarto42

Načtení a vykreslení vykreslení 2D a 3D písma, výpočet normál písma pro osvětlení, časové efekty písma, zobrazení skóre, úvodní obrazovka

1 Frames per second - počet vykreslených snímků za sekundu

Petr Dittrich – xdittr03

Fyzikální simulace: dopad míčku na desku, propad skrz díry, kolize mezi míčky, kolize s bloky

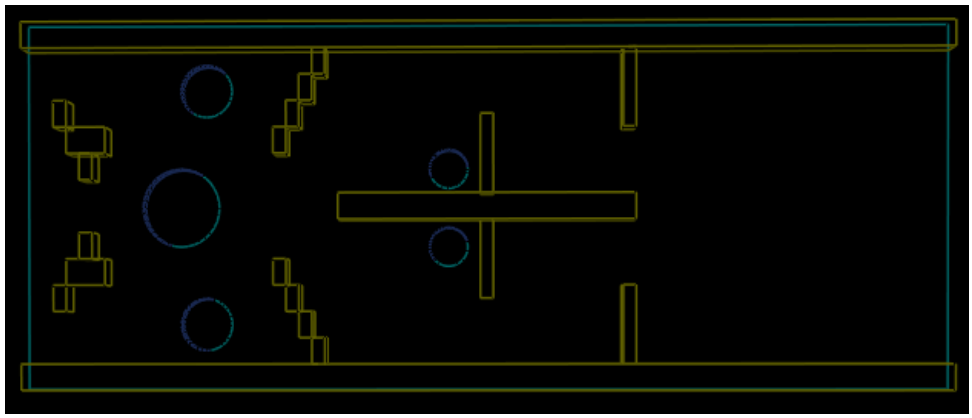
Jiří Tužil – xtuzil00

Editor úrovní, design úrovní, webová stránka hry

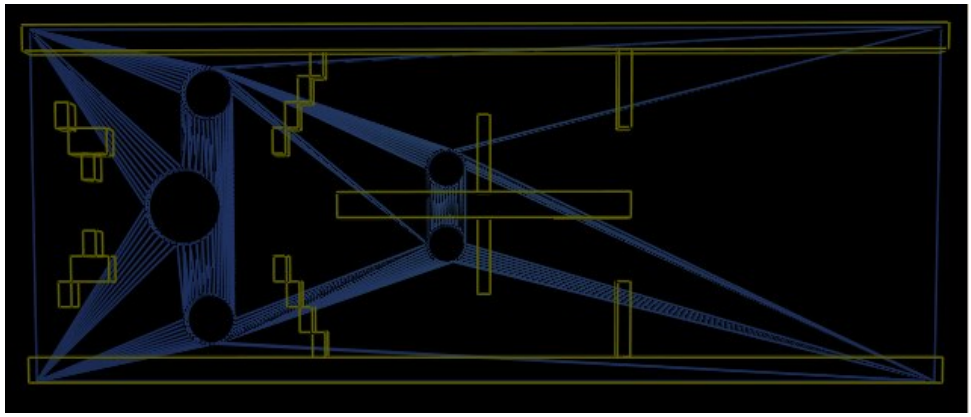
Řešené části projektu

Vykreslování scény

Jednotlivé úrovně jsou tvořeny deskou s kruhovými otvory. Implementace je taková, že vykreslíme jeden obdélníkový polygon s kruhovými otvory. Na pozici těchto otvorů je pak vykreslen také polygonový prstenec, který dává otvorům dojem hloubky.

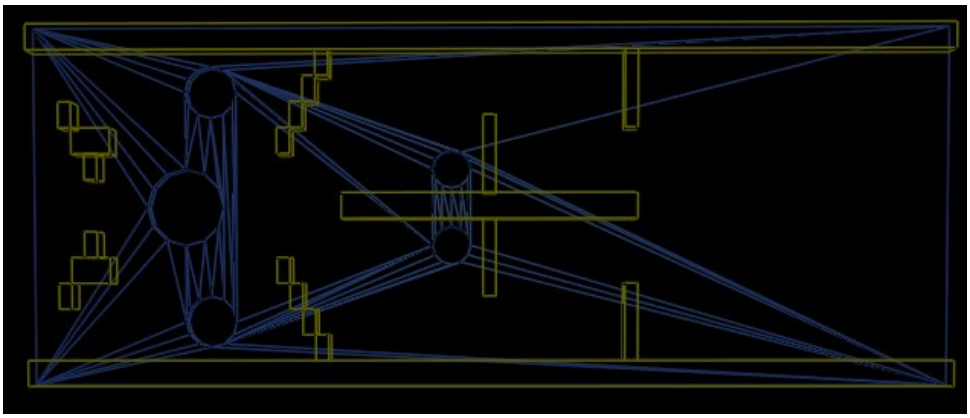


K vykreslení polygonu s otvory je třeba využít teselaci. OpenGL jsou tedy předány jednotlivé kontury na tomto polygonu, tj. obdélníkový tvar a kružnice znázorňující otvory do desky. Proces teselace pak převede zadání těchto kontur na síť trojúhelníkových polygonů, tak je vidět na následujícím obrázku.

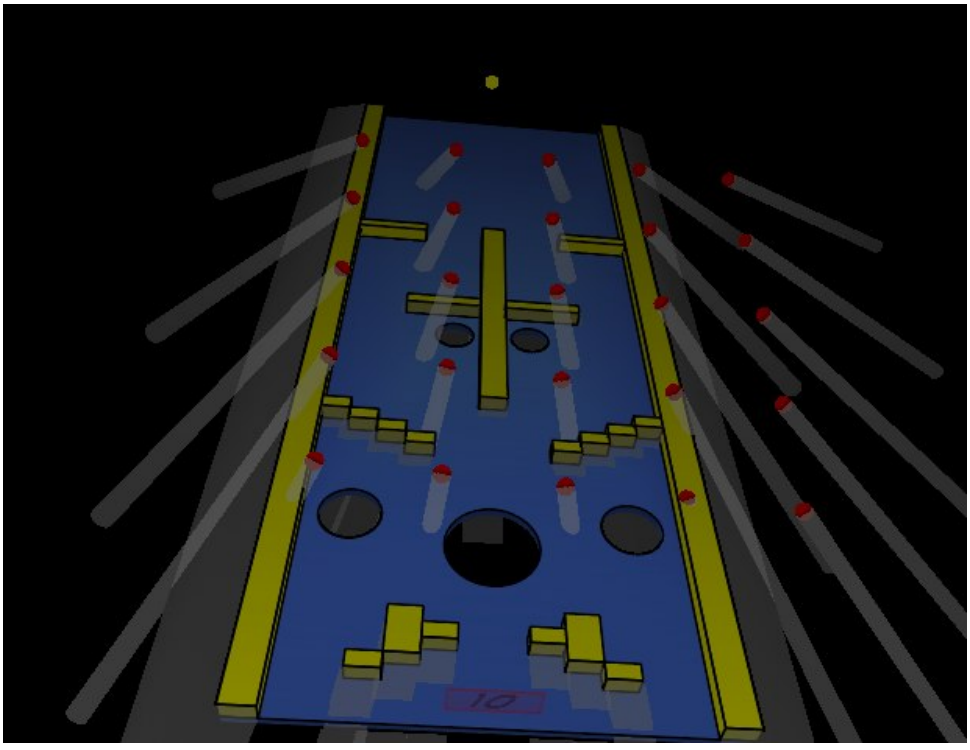


Počet polygonů ve vygenerované síti je závislý na úrovni kvality kruhových otvorů (viz kapitola Automatický konfigurátor kvality). Následující obrázek zobrazuje výsledek teselaci s třetinovou kvalitou kruhových otvorů. Počet polygonů na tomto obrázku je dramaticky nižší, což má za důsledek výrazně vyšší výkon na počítačích s menším

grafickým výpočetním výkonem. Vzhledem k tomu, že celá deska i bloky jsou v průběhu hry statické a pouze dochází k jejich otáčení, je výhodné mít tato volání předpřipravená v OpenGL *callListech*. Pokud bychom využívali teselace pro vykreslení každého snímku, došlo by k rapidnímu poklesu výkonu.



Pro nabytí dojmu prostorovosti je třeba do scény zakreslit také stíny. Booruatsukaigaumai implementuje stíny s využitím stencil bufferu. Ten umožňuje označit některé výsledné rasterizované body ještě před vykreslením. Tím lze zjistit, které body se nacházejí ve stínu a které v přímém světle. Pro označení bodů, které jsou ve stínu je nutno nejprve vykreslit stínící objekty. Ideálními stínícími objekty pro míčky jsou komolé kužely, jejichž horní hrana se míčku přímo dotýká.

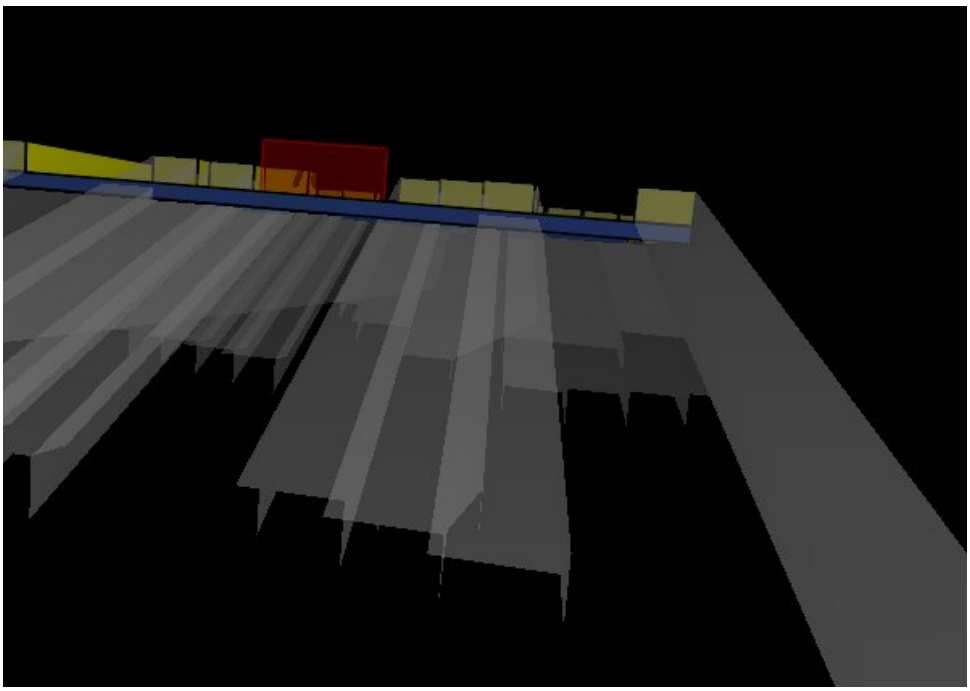


V zobrazené scéně je vykreslen bodový zdroj světla. Rotační osa všech kuželů musí při korektním výpočtu směřovat přímo do tohoto bodu.

Při vykreslování stínících objektů pro bloky je pro každou jeho hranu vygenerován stínící polygon podle vztahu

$$vertex0\,,\ vertex0+(light_{position}-vertex0)\cdot length\,,\ vertex1+(light_{position}-vertex1)\cdot length\,,\ vertex1\quad,$$

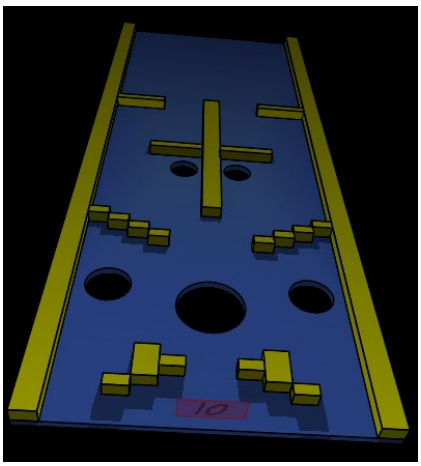
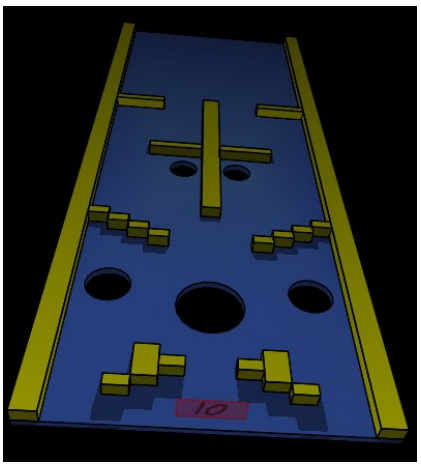
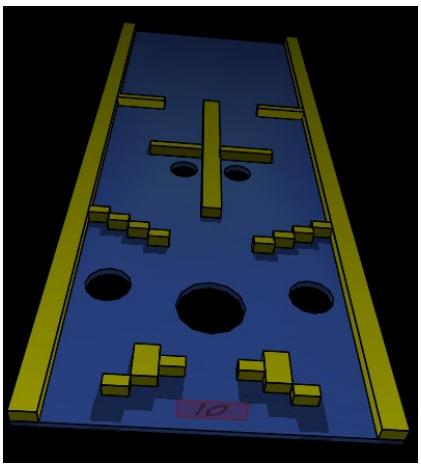
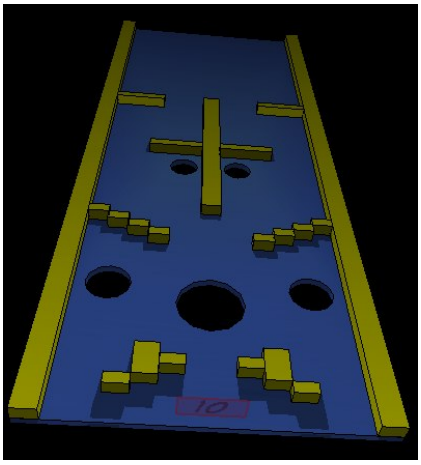
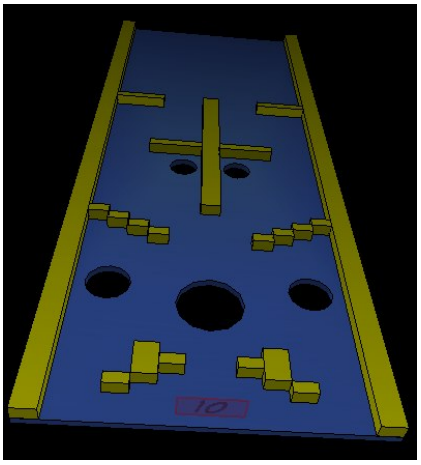
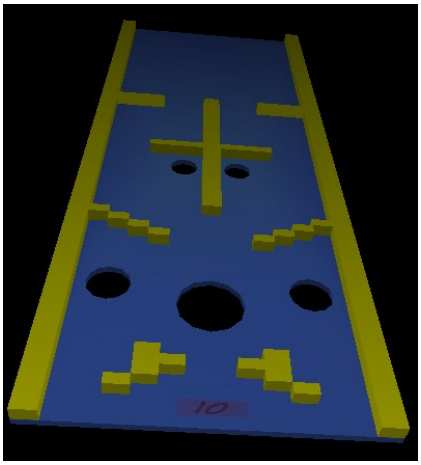
kde *length* je délka stínícího polygonu. Tato délka by měla být dostatečně velká k tomu, aby konec stínícího objektu dosahoval až pod polygon, na kterém očekáváme zobrazení stínu. Použití tohoto algoritmu opět zajišťuje, že nerovnoběžné hrany stínícího polygonu se stýkají přesně v bodu určujícím zdroj světla.



Jakmile máme označeny všechny rasterizované body, na které vyšel alespoň nějaký stínící objekt, můžeme scénu vykreslit jednou s použitím světla s jinými parametry pro místa, která mají být zastíněna a poté podruhé vykreslit místa, která mají být na přímém světle s použitím světelného zdroje s vyšší intenzitou a jasem.

Automatický konfigurátor kvality

Pokud hru spustíme na počítači, který nemá dostatečný výpočetní výkon, grafický akcelerátor nebo ovladače plně kompatibilní s grafickým akcelerátorem, je třeba snížit počet vykreslovaných polygonů a prováděných grafických operací, tak aby zůstala zachována plynulost hry na úkor kvality zobrazení. V průběhu zobrazování úrovně je průběžně počítána hodnota FPS. Zároveň je definována hranice a limitní čas pro detekci nízkého FPS. Jakmile počet snímků za sekundu klesne pod 25 a hodnota nižší než 25 snímků za sekundu je detekována po dobu delší než 1,5 sekundy je provedeno snížení detailů vždy o jeden stupeň (jednotlivé stupně kvality jsou popsány níže). Pokud ani při nižší úrovni detailů není hodnota FPS dostatečná, je provedeno další snížení. Na méně výkonných počítačích tak v ideálním případě dochází pouze k nekontinuálnímu zobrazování pouze při odpočtu před spuštěním herního kola. Samotné odpočítávání je ve hře implementováno vedle estetických důvodů právě proto, aby se včas stihla snížit kvalita vykreslování natolik, že nekontinualita vykreslování bude v herním módu zcela minimální.

		
maximální kvalita	snížení počtu polygonů u kruhových objektů na polovinu (otvory, stíny, míčky)	snížení počtu polygonů u kruhových objektů na jednu třetinu (otvory, stíny, míčky)
		
vypnutí antialiasingu čar	vypnutí stínů	vypnutí obrysů objektů

Konfigurátor a jeho účinnost byl testován na několika slabších počítačích, ale také na systému Unix s ovladači implementujícími pouze podmnožinu OpenGL, přičemž došlo k přepnutí na softwarové vykreslování.

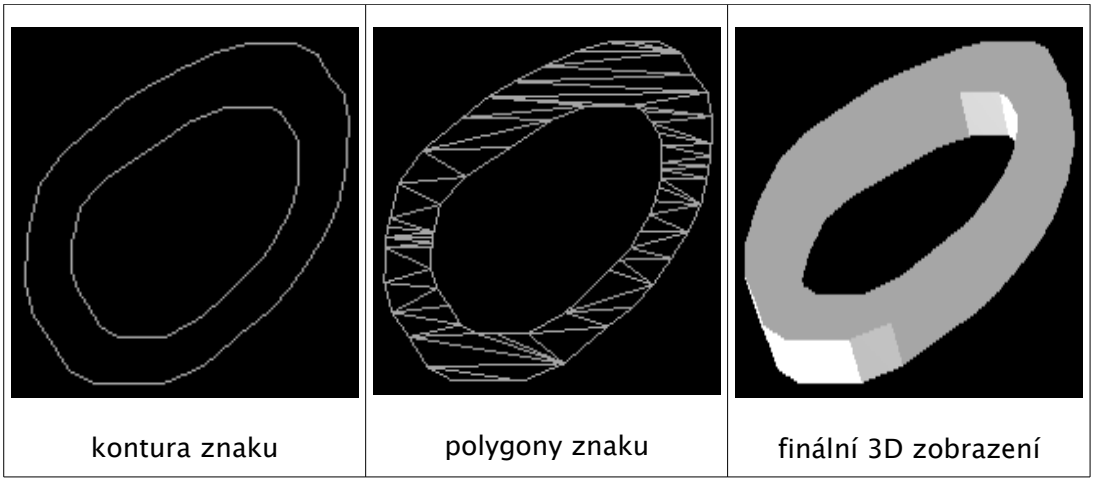
Písmo

Jelikož jsme nenašli vhodný nástroj pro vykreslování písma přímo v OpenGL ani nadstavbové knihovně GLUT, stáli jsme před problémem jak vykreslovat textové informace. Našli jsme návod [3], jak získat kontury písma a převést je na polygony.

Nejdříve pomocným programem získáme kontury zvoleného písma a uložíme je jako pole celých čísel v hlavičkovém souboru. Při spuštění hry pak tyto kontury využijeme pro vygenerování tří callListů. Jeden obsahuje vyteselované[2] [5] přední části znaku, druhý boční stěny znaku a poslední konturu znaku.

Při generování callListů pro přední a boční stěny se počítají i normálové vektory – znaky správně reagují na světelné zdroje. Tyto normálové vektory jsou nenormalizované (kvůli možnosti změnit velikosti písma) – je třeba mít povolenu automatickou normalizaci.

2 tesselace je rozložení polygonu na trojúhelníky



kontura znaku | polygony znaku | finální 3D zobrazení

Pro každý znak je také uložena jeho výška a šířka. Při vykreslování textových řetězců je pak informace o šířce znaku použita pro správný posun oproti znaku předchozímu, aby nedocházelo k jejich překrývání.

Pro vykreslování textových řetězců pak existují funkce, které vykreslí 2D znak, 3D znak[3] či pouze obrys. Také jsme zavedli funkci pro vykreslování časově omezených textů – text je zobrazen po určitý čas a během toho času se může libovolně zmenšovat či zvětšovat a také se mu postupně zvyšuje průhlednost.

Fyzikální simulace

V projektu Booruatsukaigaumai byl vytvořen fyzikální model míčku. Je zde využito jednoduché interpolace diferenciálního výpočtu diferenčním. Každý míček má uloženo své zrychlení, rychlost a pozici. Vztažná soustava pro rozložení zrychlení, rychlosti a pozice je spojena pevně s deskou, a to konkrétně s jejím středem a jejími osami. Jako zrychlení je dosazena gravitace rozdělená podle úhlu natočení desky v jednotlivých osách:

$$a_x = g \cdot \sin y \quad ,$$

$$a_y = g \cdot \sin x \quad ,$$

$$a_z = g \cdot \cos x \cdot \cos y \quad .$$

Rychlost a poloha je potom měněna o zrychlení/rychlost vynásobenou deltaT, což je časový úsek o který se posunula simulace:

$$\vec{v} = \vec{v} + \vec{a} \cdot deltaT \quad ,$$

$$\vec{pos} = \vec{pos} + \vec{v} \cdot deltaT \quad .$$

DeltaT je vypočteno jako rozdíl času a času, kdy proběhla minulá simulace. Pokud je

3 3D znak má zobrazenu pouze přední stranu a pak boční stěny, zadní strana není zobrazena

ovšem čas menší než 0.05 (tzn. rychlost klesne pod 20 fps), je nastavena deltaT na 0.05 aby nedocházelo zbytečně ke snižování hratelnosti. Pokud by tento stav byl dlouhodobý Automatický konfigurátor kvality sníží kvalitu zobrazení a simulace se vrátí na realný čas.

Všechny míčky, které opustily prostor simulace, což je krychle okolo středu desky, jsou deaktivovány.

Je provedeno zjištění, zda není míček v díře a zda je v kolizi s deskou a případně je míček od desky odražen.

Poté je testována možná kolize míčku s ostatními míčky. Pokud dojde ke kolizi, jsou míčky přisunuty k sobě, přičemž je kontrolováno, aby nebyly posunuty do desky, a je mezi nimi vyměněna rychlost.

Nakonec je testována kolize s jednotlivými bloky. Pokud dojde ke kolizi je zjištěno, se kterými stěnami byla kolize v minulém kroku a u všech ostatních je proveden odraz. Zde je navíc ošetřena možnost kolize s více bloky zároveň, například v místě jejich styku (bloků samozřejmě).

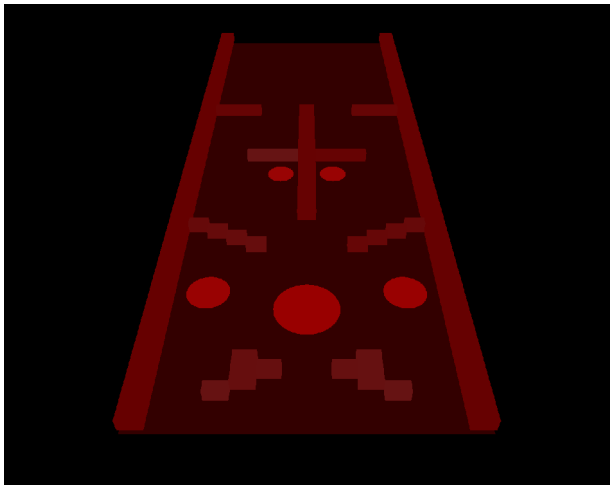
Editor levelů a uložené levely

Pro Booruatsukaigaumai byl vytvořen také WYSIWYG editor levelů. Je postaven nad vykreslovacím jádrem hry, to znamená, že tak, jak scénu při editaci vidíme, tak bude později vypadat ve hře.

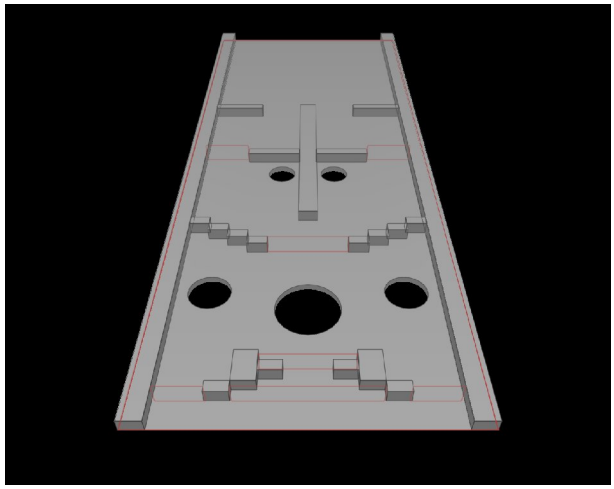
Editor není nedílnou součástí hry a s jeho distribucí se nepočítá. Z toho důvodu není jeho ovládání zrovna intuitivní, přestože je snadno osvojitelné. Výběr objektů se provádí klávesami Page Up / Page Down / Home / End nebo ukázáním myší. Vybraný objekt je ve scéně označen červeným rámečkem. Pro posuny a změny velikostí objektů pak slouží šipky a klávesy + / - / * / / spolu s přepínačem Shift pro změnu velikosti kroku na desetinu normálního. Tlačítky 0 – 9 se přepíná aktuálně editovaný level, klávesou Esc se uloží změny a editor ukončí.

Načítání a ukládání levelů probíhá přímo z/do C zdrojového kódu. Výhodou je nulová režie a jednosouborový design hry, nevýhodou nutnost rekompilace editoru i hry po každé změně v levelech. Se změnami se ale po ukončení vývoje nepočítá.

Zajímavostí je realizace výběru objektů myší. Celé scéna je nejprve vykreslena v jednolitých barvách bez stínování do druhé stránky framebufferu. Každý objekt je vykreslen jinou barvou se zachováním funkce Z-bufferu. Barva pixelu, nad kterým je umístěn kurzor myši pak jednoznačně identifikuje objekt, na který myš ukazuje a můžeme s ním dále pracovat. Přitom stránka framebufferu s vykreslenou plochou scénou není vůbec přepnuta a vykreslena, takže uživatel vidí stále správně barevnou stínovanou scénu bez „blikání“.



nestínovaná scéna pro myšování



standardní stínované prostředí editoru

Použité zdroje

[1] Tišnovský P., Dobšík M., Herout A.: Cvičení číslo 3 do předmětu Počítačová grafika - *kostra modulu main, stíny pomocí stencil bufferu*

https://www.fit.vutbr.cz/study/courses/PGR/private/lab03/lab03_04.c

[2] Diskuzní fórum GameDev.Net - *kód pro rotaci míčku simulující kutálení*

http://www.gamedev.net/community/forums/topic.asp?topic_id=190307

[3] Michael Fötsch: Triangulating and Extruding Arbitrary Polygons - *generování kontur písmen a vytažení 3D písma*

http://www.geocities.com/foetsch/extrude/extrude.htm

[4] Microsoft Developer Network: Using Paths - *generování kontur písem*

http://msdn2.microsoft.com/en-us/library/ms535174.aspx

[5] Pavel Tišnovský: OpenGL a nadstavbová knihovna GLU, 19. díl;

http://www.root.cz/clanky/opengl-a-nadstavbova-knihovna-glu-19/