

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Vykreslování Krychlí ve 3D Prostředí
Tabulkového Procesoru

Ondřej Čopák

Hlavní město Praha

Hlavní město Praha, 2024

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Vykreslování Krychlí ve 3D Prostředí Tabulkového Procesoru

Rendering Cubes in the 3D Environment of a Spreadsheet

Jméno: Ondřej Čopák

Škola: Gymnázium Christiana Dopplera, Zborovská 621/45, Praha

Kraj: Hlavní město Praha

Konzultant: Bez konzultanta

Hlavní město Praha, 2024

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů. Veškeré obrázky jsou vlastní tvorby.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne 26. března 2024

Ondřej Čopák

Poděkování

Chtěl bych poděkovat svým kolegům a především svému učiteli Bc. Petru Vincenovi za pomoc s formální a odbornou částí práce. Dále bych chtěl poděkovat internetové komunitě portování počítačové hry Doom na různé herní enginy, převážně moderátorovi Wojtkovi Grajovi, za podporu a konzultaci ohledně teorie raycastingu.

Abstrakt

Práce se zabývá zobrazováním otexturovaných krychlí v tabulkovém prostředí Microsoft Excel. Pro účely této práce jsem implementoval v jazyce VBA program, který dokáže vykreslovat krychle v závislosti na pozici a natočení kamery hráče. Dále se práce zaměřuje na funkci Excelu jako herního enginu a matematickou část zobrazování z 3D do buněk pomocí perspektivní projekce. V závěru se zabývám optimalizací programu tak, aby se jednotlivé snímky vykreslovaly co nejrychleji. Výsledkem je schopnost vykreslit snímek v přijatelném čase.

Klíčová slova

Perspektivní projekce; 3D prostředí; Microsoft Excel; Textury; Optimalizace

Abstract

The thesis is about displaying textured cubes in the spreadsheet environment of Microsoft Excel. For this purpose, I have developed a VBA program capable of rendering cubes according to the position and rotation of the player's camera. Additionally, this thesis focuses on the function of Excel as a game engine and the mathematical aspects of projecting 3D cubes onto cells using perspective projection. In conclusion, emphasis is placed on optimizing the program to achieve faster rendering of individual frames. The result is the ability to render a frame in an acceptable time.

Keywords

Perspective projection, 3D environment; Microsoft Excel; Textures; Optimization

OBSAH

1	Úvod	4
2	Teorie	5
2.1	Posun ve 3D	5
2.2	Rotace ve 3D	6
2.2.1	Odvození vzorce pro otáčení ve 2D	6
2.2.2	Rotace kolem osy Y	10
2.2.3	Rotace kolem osy X (pitch)	11
2.3	Transformační matice	12
2.4	Zobrazení do 2D	13
3	Implementace	15
3.1	Prostředí Excelu	15
3.2	VBA	15
3.3	Uživatelské rozhraní	16
3.3.1	Horní lišta	16
3.3.2	Player position	16
3.3.3	Camera orientation	17
3.3.4	Stats	17
3.3.5	Variables	17
3.3.6	Blocks	17
3.3.7	Texture List	17
3.3.8	Timestamps	18
3.4	Třídy (Classes)	18
3.4.1	Herní prostředí	18
3.4.1.1	Player	18
3.4.1.2	Game	19
3.4.1.3	Calculations	19
3.4.1.4	Textures	19
3.4.1.5	Stats	19
3.4.2	Geometrické objekty	20
3.4.2.1	Block	20
3.4.2.2	Side	20
3.4.2.3	Pixel	20
3.5	Procedury (Sub, Functions)	20

3.5.1	Main	21
3.5.1.1	Init (Sub)	21
3.5.1.2	Move (Sub)	21
3.5.1.3	CalculateSides (Function)	21
3.5.1.4	ApplyTexture (Function)	21
3.5.1.5	ConvertDraw2D (Function)	23
3.5.2	Geometry	24
3.5.2.1	CalculateCoordinates (Function)	24
3.5.2.2	IsPointInsideFOV (Function)	24
3.5.2.3	GetLinePixels (Sub)	25
3.5.3	Functions	26
3.5.3.1	RemoveDuplicateSides (Sub)	26
3.5.3.2	ReverseCollection (Function)	26
3.5.3.3	SortByDistance (Function)	26
3.5.3.4	QuickSort (Sub)	26
3.5.4	Visual	26
3.5.4.1	ClearScreen (Sub)	27
3.5.4.2	FillCellsInRange (Sub)	27
3.5.4.3	SetPlayer/Variables (Sub)	27
3.5.5	Keys (Ovládání)	27
3.5.5.1	BindKeys (Sub)	27
3.5.5.2	FreeKeys (Sub)	28
3.5.5.3	MoveTop/Bottom (Sub)	28
3.5.5.4	MoveLeft/Right (Sub)	28
3.5.5.5	MoveFront/Back (Sub)	29
3.5.5.6	LookTop/Bottom (Sub)	29
3.5.5.7	LookLeft/Right (Sub)	30
3.6	Možnosti krychlí	30
3.6.1	Krychle	30
3.6.2	Půlkrychle	30
3.6.3	Přidávání jiných tvarů	31
3.6.4	Průhlednost	31
4	Optimalizace	32
4.1	Limitace VBA	32
4.2	Složitosti funkcí	32
4.2.1	Init	32
4.2.2	CalculatePosition	33
4.2.3	ApplyTexture	33
4.2.4	ConvertDraw2D	33
4.2.5	Celková	33
4.3	Implementace optimalizací	34

4.3.1	Závislost výkonu	34
4.3.1.1	Počet krychlí	34
4.3.1.2	Počet buněk	35
4.3.2	Sady testů	36
4.3.3	Bez optimalizace	36
4.3.4	Konkrétní optimalizace	37
4.3.4.1	Zorné pole hráče	37
4.3.4.2	Počet stran krychle	37
4.3.4.3	Duplicitní strany	37
4.3.4.4	Vykreslování po řádcích	38
4.3.5	Veškeré optimalizace	38
4.3.6	Další možnosti	38
4.4	Shrnutí struktury programu	40
5	Diskuse	41
5.1	Optimalizace	41
5.1.1	GPU	41
5.1.2	Zakreslování buněk	42
5.2	Využití	44
5.2.1	Edukační	44
5.2.2	3D engine	44
5.2.3	Animace	44
5.3	Možnosti pokračování	45
6	Závěr	47
	Literatura	48
	Seznam pojmů	50
	Seznam obrázků	51
	Seznam tabulek	53
	Seznam ukázek kódů	54
	Přílohy	55

1 ÚVOD

Cílem této práce je přiblížit základní principy vykreslování 3D objektů v herních enginech. Toto téma jsem si vybral s cílem rozšířit dosah celosvětového trendu tzv. *portování* (starších) her, jako například Doom¹, do různých prostředí. Rozhodl jsem se tak implementovat základy grafického zobrazování krychlí, inspirované počítačovou hrou Minecraft², do prostředí Microsoft Excelu³ (dále jen *Excel*).

Důraz je také kladen na vysvětlení, jak Excel může být vnímán jako prostředí pro programování 3D her. V rámci této práce se analyzuje otáčení kamery, pohybování hráče, převod do 2D a další aspekty týkající se vizualizací. Velkou částí je také optimalizace kódu tak, aby se minimalizoval počet výpočtů, s cílem zrychlit program. Předpokladem pro dobrou orientaci v práci je základní porozumění fungování počítačové grafiky.

Celá tato práce je rozdělena do tří hlavních částí (teoretický základ, implementace a optimalizace) a je dále členěna do podkapitol. Teoretická část se věnuje algoritmům a matematice, která stojí za pohybováním v třídimenzionálním prostoru. Druhá část se zaměřuje na konkrétní implementaci do prostředí Excelu i na uživatelskou interakci s programem, a v poslední části se práce zabývá limitujícími faktory prostředí a dalšími optimalizacemi kódu. V úplném závěru (a diskusi) pak práce rozebírá různé možnosti vylepšení programu i jeho konkrétní využití.

Po celou práci jsem si stanovil 5 bodů, kterými se má program řídit

- **Srozumitelnost** – Kód by měl být co nejvíc srozumitelný, vysvětlen a okomentován.
- **Rychlost** – Výsledné vykreslování by mělo být v rámci možností co nejrychlejší.
- **Ovládání** – Kontrolování kamery, hráče a nastavení by mělo být co nejjednodušší.
- **Vše ve VBA** – Celý program musí být napsaný v jazyce VBA (v prostředí Excelu).
- **Jednoduché na modifikace** – Pro další vývojáře by mělo být snadné pochopit, jak přidávat nové textury, objekty, provádět optimalizace a další modifikace.

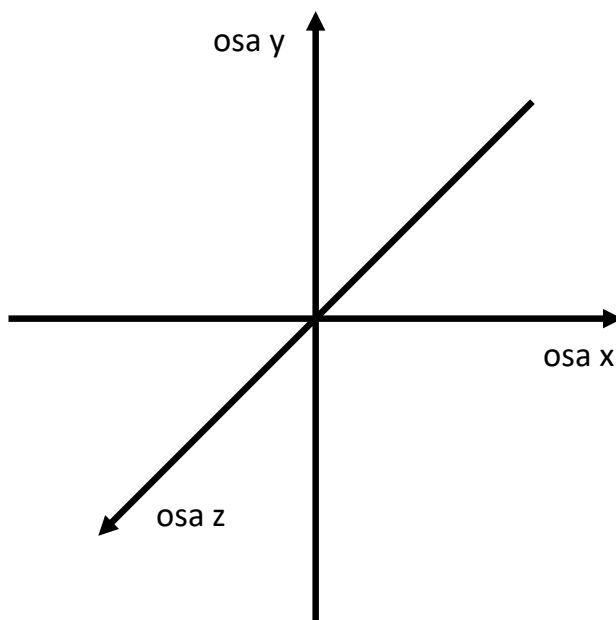
¹Doom (1993, id Software)

²Minecraft (2009, Mojang Studios)

³Microsoft Excel (2016, Microsoft)

2 TEORETICKÝ ZÁKLAD 3D PROSTŘEDÍ

Kapitola se věnuje teoretickému základu matematických operací – pohybu, otáčení ve 3D prostředí a následnému převodu do 2D v lineárním zobrazení. Osy X , Y a Z jsou v průběhu celé práce vždy pravotočivé (obr. 2.1). Tvoří tzv. *kartézský souřadnicový systém*, který určuje jednoznačné souřadnice bodů [1].



obr. 2.1: pravotočivé osy X, Y, Z

2.1 POSUN VE 3D

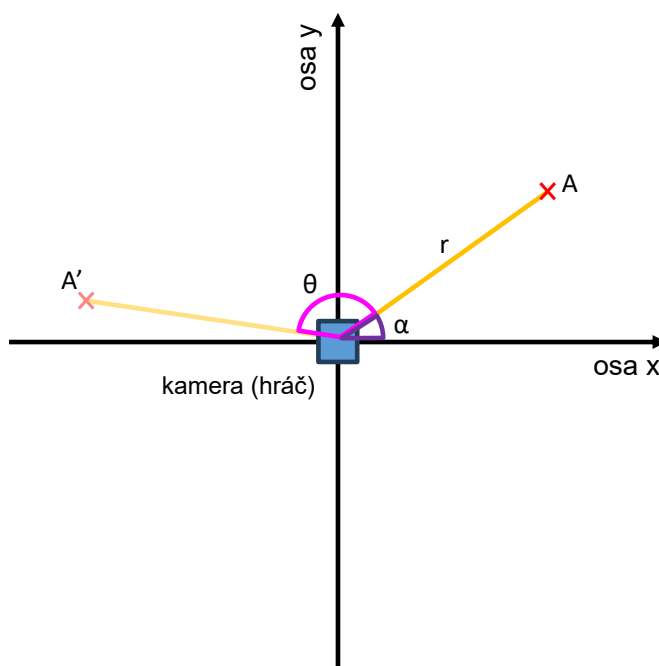
Pro minimalizaci složitosti výpočtů je výhodné umístit hráče do počátečního bodu souřadnicového systému, konkrétně do bodu $[0, 0, 0]$. Tímto se eliminuje nutnost současné práce s pozicí hráče a polohou krychlí a zjednoduší se tím výpočty. Každá krychle se proto posune po jednotlivých osách dle původního umístění hráče. Místo pohybování hráče se tedy bude pohybovat svět (o opačné hodnoty x, y, z)

2.2 ROTACE VE 3D

Kamera, podobně jako hráč, se také otočí tak, aby byla orientovaná výchozím směrem $[0, 0, 0]$. Provádí se tedy rotace všech bodů tak, aby zůstaly ve stejné vzdálenosti od hráče, ale pod daným úhlem kamery. Program umožňuje rotaci kamery pouze ve dvou směrech – *nahoru/dolů* a *doleva/doprava*. V původní hře nelze naklánět hlavu směrem k ramenům. Pro lepší vizualizaci se pracuje v této části práce ve 2D rovině, kde hráč je v bodě $[0, 0]$ a zobrazují se pouze osy, ve kterých dochází ke změně hodnot během konkrétního otočení.

2.2.1 Odvození vzorce pro otáčení ve 2D

Pro přesné umístění a orientaci objektů je nezbytné odvodit vzorec pro otáčení ve 2D rovině. Tento vzorec umožní vypočítat nové souřadnice bodu po rotaci kolem počátku. Cílem je vypočítat nové souřadnice bodu $[x, y]$ na základě úhlu rotace (obr. 2.2).



obr. 2.2: otáčení bodu A o úhel θ v kartézské soustavě souřadnic roviny XY

A: bod určen souřadnicemi $[x, y]$, který je otáčen.

A': výsledný bod po otáčení.

θ : úhel rotace (yaw, pitch).

α : původní úhel mezi osou X a bodem.

β : celkový úhel rotace ($\alpha + \theta$).

r: vzdálenost od počátku k bodu ve 2D rovině.

Pro lepší pochopení je zapotřebí odvodit vzorec pro otáčení ve 2D na základě (obr. 2.2) Úhel α je dán arkus tangem podílu y/x a je dán také kvadrantem, ve kterém se bod nachází.

$$\alpha = \arctan\left(\frac{y}{x}\right)$$

Z toho lze spočítat celkový úhel rotace bodu A .

$$\beta = \alpha + \theta$$

Nové souřadnice bodu proto jsou:

$$x' = r \cdot \cos(\beta)$$

$$y' = r \cdot \sin(\beta)$$

Abychom se vyhnuli problému spojeným s dělením nulou (v případě $x = 0$) a zohlednili kvadrant, ve kterém se souřadnice nacházejí, je třeba upravit vzorce tak, aby obě souřadnice byly vyjádřené pomocí obou goniometrických funkcí $\sin(\theta)$ a $\cos(\theta)$. Lze proto začít úpravy použitím součtového vzorce pro $\tan(\alpha + \theta)$.

$$\tan(\alpha + \theta) = \frac{\tan(\alpha) + \tan(\theta)}{1 - \tan(\alpha) \cdot \tan(\theta)}$$

Hodnota $\tan(\arctan(\frac{y}{x}))$ se rovná $\frac{y}{x}$ a β se rovná $\alpha + \theta$.

$$\tan(\beta) = \frac{\frac{y}{x} + \tan(\theta)}{1 - \frac{y}{x} \cdot \tan(\theta)}$$

Dále se zlomek rozšíří o výraz $x \cdot \cos(\theta)$.

$$\tan(\beta) = \frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}$$

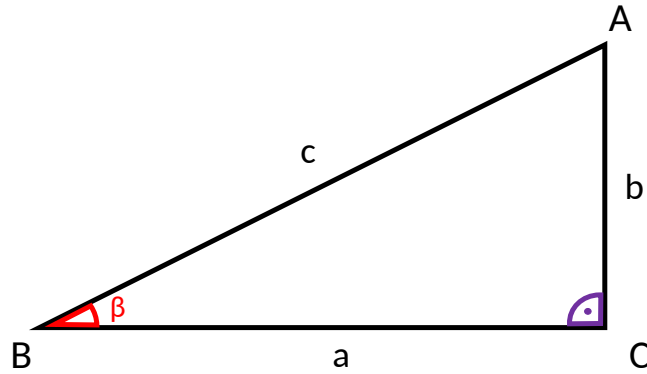
Následně lze vyjádřit β z této rovnice.

$$\beta = \arctan\left(\frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}\right)$$

Následuje dosazení β z předchozího kroku do vzorců.

$$x' = r \cdot \cos\left(\arctan\left(\frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}\right)\right)$$

$$y' = r \cdot \sin\left(\arctan\left(\frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}\right)\right)$$

obr. 2.3: pravoúhlý trojúhelník s úhlem β u vrcholu B

Pro dokončení úprav je nutné spočítat hodnoty $\cos(\arctan(\frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}))$ a $\sin(\arctan(\frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}))$

Z předchozích výpočtů:

$$\beta = \arctan\left(\frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}\right)$$

A zároveň musí pro obecný úhel θ platit (obr. 2.3):

$$\tan(\beta) = \frac{b}{a}$$

Toto lze převést na:

$$\tan(\beta) = \tan\left(\arctan\left(\frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}\right)\right) = \frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}$$

Z obou rovnic vyplývá, že

$$\frac{b}{a} = \frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{x \cdot \cos(\theta) - y \cdot \sin(\theta)}$$

Můžeme obě strany a , b vyjádřit z tohoto poměru.

$$b = y \cdot \cos(\theta) + x \cdot \sin(\theta)$$

$$a = x \cdot \cos(\theta) - y \cdot \sin(\theta)$$

Funkce $\sin(\beta)$ a $\cos(\beta)$ vyžadují stranu c k dopočítání. Tu stranu lze vyjádřit pomocí Pythagorovy věty z (obr. 2.3) jako $c = \sqrt{a^2 + b^2}$.

$$\sin(\beta) = \frac{b}{c} = \frac{b}{\sqrt{a^2 + b^2}}$$

$$\cos(\beta) = \frac{a}{c} = \frac{a}{\sqrt{a^2 + b^2}}$$

Po dosazení za a , b a do původní soustavy rovnic se to upraví na:

$$x' = r \cdot \frac{x \cdot \cos(\theta) - y \cdot \sin(\theta)}{\sqrt{(y \cdot \cos(\theta) + x \cdot \sin(\theta))^2 + (x \cdot \cos(\theta) - y \cdot \sin(\theta))^2}}$$

$$y' = r \cdot \frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{\sqrt{(y \cdot \cos(\theta) + x \cdot \sin(\theta))^2 + (x \cdot \cos(\theta) - y \cdot \sin(\theta))^2}}$$

Roznásobení závorek.

$$x' = r \cdot \frac{x \cdot \cos(\theta) - y \cdot \sin(\theta)}{\sqrt{y^2 \cos(\theta)^2 + xy \cos(\theta) \sin(\theta) + x^2 \sin(\theta)^2 + x^2 \cos(\theta)^2 - xy \cos(\theta) \sin(\theta) + y^2 \sin(\theta)^2}}$$

$$y' = r \cdot \frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{\sqrt{y^2 \cos(\theta)^2 + xy \cos(\theta) \sin(\theta) + x^2 \sin(\theta)^2 + x^2 \cos(\theta)^2 - xy \cos(\theta) \sin(\theta) + y^2 \sin(\theta)^2}}$$

Odečtení $x \cdot y \cos(\theta) \sin(\theta)$.

$$x' = r \cdot \frac{x \cdot \cos(\theta) - y \cdot \sin(\theta)}{\sqrt{y^2 \cos(\theta)^2 + x^2 \sin(\theta)^2 + x^2 \cos(\theta)^2 + y^2 \sin(\theta)^2}}$$

$$y' = r \cdot \frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{\sqrt{y^2 \cos(\theta)^2 + x^2 \sin(\theta)^2 + x^2 \cos(\theta)^2 + y^2 \sin(\theta)^2}}$$

Vytknutí x^2 a y^2 .

$$x' = r \cdot \frac{x \cdot \cos(\theta) - y \cdot \sin(\theta)}{\sqrt{x^2 \cdot (\sin(\theta)^2 + \cos(\theta)^2) + y^2 \cdot (\sin(\theta)^2 + \cos(\theta)^2)}}$$

$$y' = r \cdot \frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{\sqrt{x^2 \cdot (\sin(\theta)^2 + \cos(\theta)^2) + y^2 \cdot (\sin(\theta)^2 + \cos(\theta)^2)}}$$

Úprava podle $\sin(\theta)^2 + \cos(\theta)^2 = 1$.

$$x' = r \cdot \frac{x \cdot \cos(\theta) - y \cdot \sin(\theta)}{\sqrt{x^2 + y^2}}$$

$$y' = r \cdot \frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{\sqrt{x^2 + y^2}}$$

Vyjádření r jako vzdálenost bodu od počátku souřadnic.

$$x' = \sqrt{x^2 + y^2} \cdot \frac{x \cdot \cos(\theta) - y \cdot \sin(\theta)}{\sqrt{x^2 + y^2}}$$

$$y' = \sqrt{x^2 + y^2} \cdot \frac{y \cdot \cos(\theta) + x \cdot \sin(\theta)}{\sqrt{x^2 + y^2}}$$

Zkrácení a úprava na finální tvar:

$$x' = x \cdot \cos(\theta) - y \cdot \sin(\theta)$$

$$y' = x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

V obou rotacích budou tyto vzorce využity.

2.2.2 Rotace kolem osy Y (yaw)

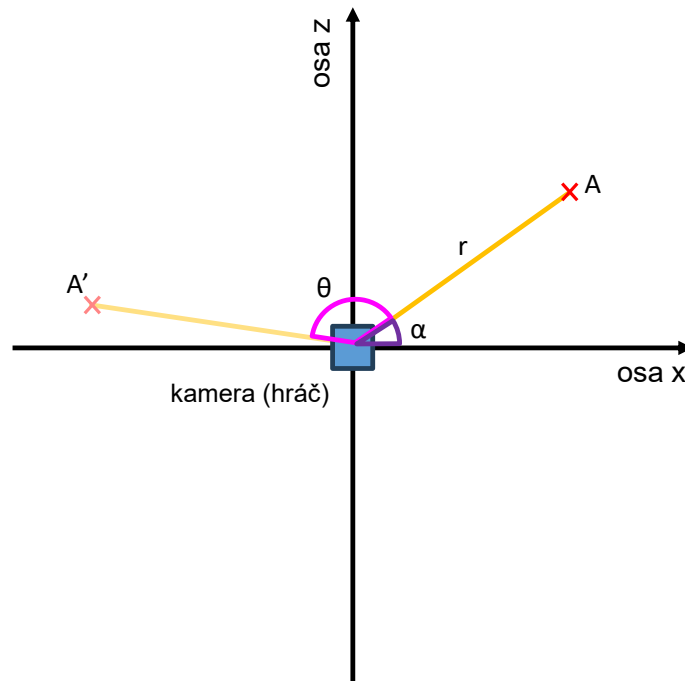
Pro rotaci roviny XZ (otáčení kamery doleva a doprava) lze využít již spočítaný vzorec, kde Y zůstane stejné a X a Z se mění v závislosti na úhlu yaw (obr. 2.4). Použijí cyklickou záměnu vzorce z oddílu 2.2.1.

θ je úhlem yaw .

$$x' = x \cdot \cos(\theta) - z \cdot \sin(\theta)$$

$$y' = y$$

$$z' = x \cdot \sin(\theta) + z \cdot \cos(\theta)$$



obr. 2.4: otáčení bodu A o úhel θ (yaw) v případě kamery otáčení kolem osy Y

2.2.3 Rotace kolem osy X (pitch)

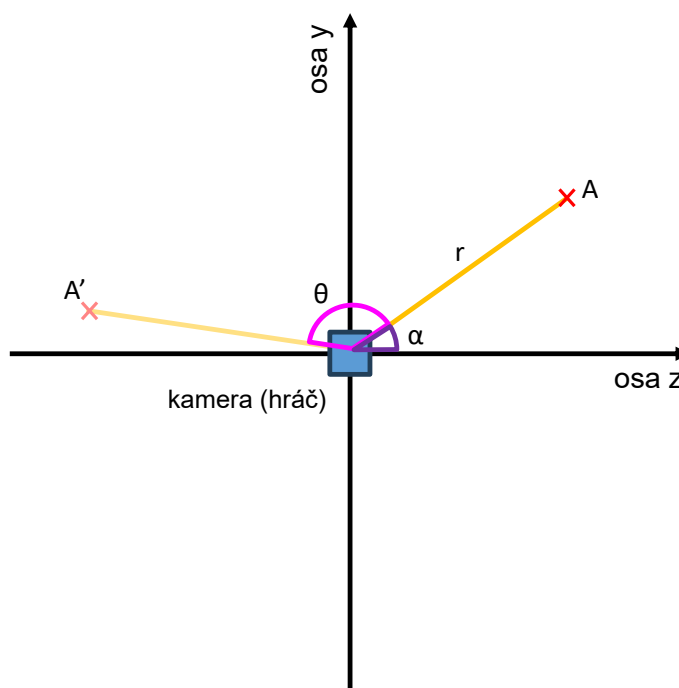
Pro rotaci roviny YZ (otáčení kamery nahoru a dolů) opět použijí odvozený vzorec. Tentokrát cyklickou záměnou zůstane stejná souřadnice X , kde hodnoty Y a Z se opět mění v závislosti na úhlu *pitch* (obr. 2.5).

θ odpovídá úhlu *pitch*.

$$x' = x$$

$$y' = y \cdot \cos(\theta) - z \cdot \sin(\theta)$$

$$z' = y \cdot \sin(\theta) + z \cdot \cos(\theta)$$



obr. 2.5: otáčení bodu A o úhel θ (*pitch*) v případě kamery otáčení kolem osy X

2.3 TRANSFORMAČNÍ MATICE VE 3D

V programu (kód 2.3) se souřadnice jednotlivých bodů počítají pomocí rovnic odvozených v předchozích částech práce (oddíly 2.1, 2.2), protože jazyk VBA není optimalizovaný pro práci s maticemi (oddíl 4.1).

```

1  ' Translate the point relative to the player's position
2  transVertex(0) = point(0) - P.x
3  transVertex(1) = point(1) - P.y
4  transVertex(2) = point(2) - P.z
5
6  ' Rotate the point around the player's yaw (axe Y)
7  rotVertexY(0) = transVertex(0) * c.cos(P.yaw) - transVertex(2) * c.sin(P.yaw)
8  rotVertexY(1) = transVertex(1)
9  rotVertexY(2) = transVertex(0) * c.sin(P.yaw) + transVertex(2) * c.cos(P.yaw)
10
11 ' Rotate the point around the player's pitch (axe X)
12 rotVertexP(0) = rotVertexY(0)
13 rotVertexP(1) = rotVertexY(1) * c.cos(P.pitch) - rotVertexY(2) * c.sin(P.pitch)
14 rotVertexP(2) = rotVertexY(1) * c.sin(P.pitch) + rotVertexY(2) * c.cos(P.pitch)

```

kód 2.1: perspektivní zobrazení bodů; názvy proměnných upraveny

Pohyb i rotaci kamery lze kombinovat a vyjádřit pomocí tzv. Transformačních Matic. Lze proto tyto odvozené rovnice převést na matice. Výsledná pozice bodu po aplikování posunu i obou rotací bude odpovídat součinu těchto matic, protože otočení kolem dvou různých os lze převést pomocí skládání rotací na otáčení nejprve kolem první osy a následně kolem druhé.

Použití matic může výrazně urychlit výpočty, protože se nejprve vynásobí veškeré transformační matice s rotačními a následně se tato jedna matice aplikuje na všechny body.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} c \cdot \cos(P.\text{yaw}) & 0 & -c \cdot \sin(P.\text{yaw}) \\ 0 & 1 & 0 \\ c \cdot \sin(P.\text{yaw}) & 0 & c \cdot \cos(P.\text{yaw}) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c \cdot \cos(P.\text{pitch}) & -c \cdot \sin(P.\text{pitch}) \\ 0 & c \cdot \sin(P.\text{pitch}) & c \cdot \cos(P.\text{pitch}) \end{bmatrix} \begin{bmatrix} x - P.x \\ y - P.y \\ z - P.z \end{bmatrix}$$

(transformační matice 3D bodu pro otáčení yaw, pitch a translaci)

Pořadí rotací a translace je nezaměnitelné. To vyplývá z vlastnosti nekomutativnosti matic. V opačném případě by došlo k vypočítání jiných souřadnic.

2.4 ZOBRAZENÍ DO 2D ROVINY

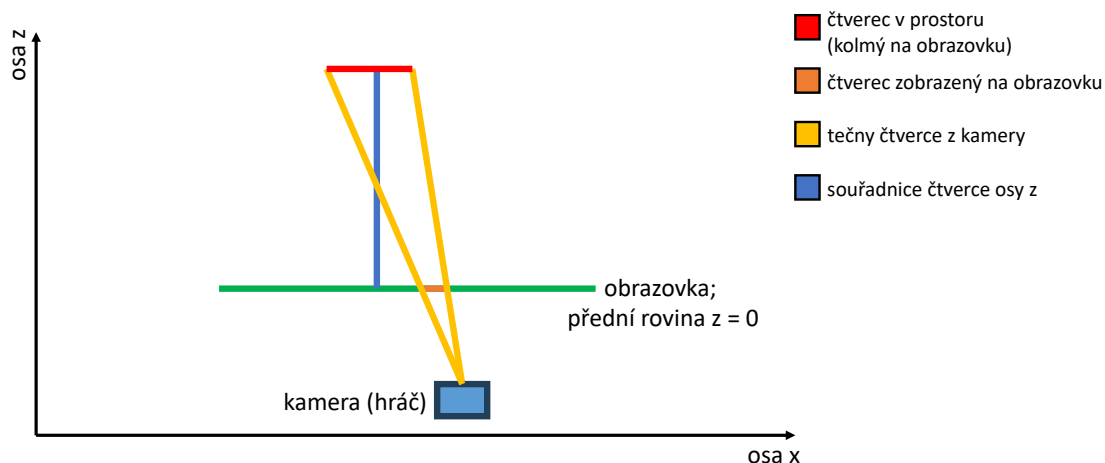
Při převodu 3D prostoru do 2D roviny je využíván princip *perspektivní projekce*. To zajišťuje hloubkovou vizualizaci prostorových objektů na plochu obrazovky [2].

Perspektivní projekce pracuje s tzv. normalizovanými souřadnicemi, jejichž princip je vysvětlen v předchozí části práce (viz oddíl 2.1 a 2.2). Normalizované souřadnice x, y a z , získané při pohledu hráče ve směru osy z , lze následně převést na odpovídající 2D souřadnice obrazovky.

$$X_{2D} = \frac{X_{3D}}{Z_{3D}}, \quad Y_{2D} = \frac{Y_{3D}}{Z_{3D}}$$

Tyto vzorce využívají podobnost trojúhelníků (*červená strana* s kamerou a *oranžová strana* s kamerou), kde *oranžová strana* vznikla jako úsečka mezi body, které jsou průsečíky tečen (z kamery k původnímu *červenému* čtverci) s rovinou obrazovky [3]. Koeficient podobnosti vychází ze vzdálenosti od roviny XY (souřadnice z) (obr. 2.6).

S růstem vzdálenosti stran krychle od obrazovky bude její obsah (zobrazeného útvaru na obrazovce) klesat. Program tedy počítá pozice na základě podílů v osách obrazovky s vzdáleností souřadnice z . Pokud krychle prochází obrazovkou, tak se bod posune 1 jednotku ve směru po ose z , aby nevznikl problém s dělením nulou a uživatel mohl vidět danou stranu na obrazovce (kód 2.4; řádek 2).



obr. 2.6: perspektivní zobrazení čtverců na obrazovku, pohled shora – převedeno do 2D, pro lepší orientaci

```
1  If intersectionPoint(2) == 0 Then
2      intersectionPoint = Array(intersectionPoint(0), intersectionPoint(1), 1)
3  End If
4
5  projectedX = intersectionPoint(0) / intersectionPoint(2)
6  projectedY = intersectionPoint(1) / intersectionPoint(2)
```

kód 2.2: perspektivní zobrazení bodů; upraveno

3 IMPLEMENTACE PROGRAMU

V této části se zaměřím na konkrétní implementaci celého programu v prostředí Excelu s využitím jazyka Visual Basic for Applications (dále jen *VBA*). Následně detailně rozeberu klíčové algoritmy, funkce a třídy, nezbytné pro funkci programu. Program je napsán v anglickém jazyce, aby byl dostupný co nejširšímu spektru uživatelů. Zdrojový kód spolu s komentáři je k dispozici v příloze práce.

3.1 PROSTŘEDÍ EXCELU

V Excelu plní buňky roli pixelů, které se organizují do tabulky, fungující jako obrazovka, kde vytvářejí výsledný obrázek (*snímek*).

Excelový soubor se skládá ze tří listů. První list slouží jako hlavní obrazovka, na níž se zobrazuje herní scéna. Druhý list slouží pro nastavení, výpis statistik a další ovládací prvky. Poslední list slouží k uchování všech textur. Každá textura je vždy rozdělena do tří skupin buněk o rozměrech 8x8. První skupina obsahuje vrchní texturu krychle, prostřední představuje všechny boční strany krychle a třetí skupina zahrnuje spodní texturu krychle.

3.2 VBA (VISUAL BASIC FOR APPLICATIONS)

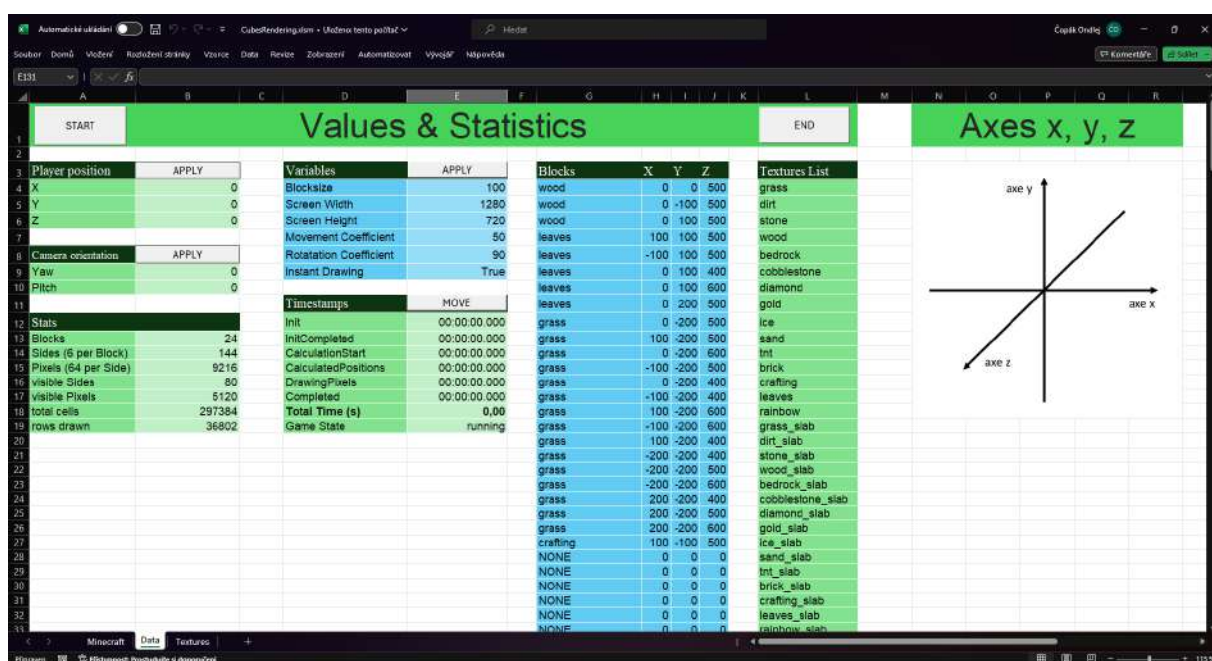
Pro implementaci programu jsem zvolil programovací jazyk VBA, jelikož je to jediný jazyk, který Excel podporuje. Tato volba má svá pozitiva – veškerá dokumentace k funkcím je vytvořena pouze pro tento jazyk, ale má také svá negativa, protože uživatel nemá možnost vybrat si lépe optimalizovaný jazyk.

VBA patří mezi interpretované jazyky [4]. Kód napsaný ve VBA je přeložen do proprietárního prováděcího kódu nazvaného *Microsoft P-code* a poté je ukládán jako zdrojový kód v dokumentu Excelu [4]. VBA umožňuje například ovládání Excelu a manipulaci s buňkami, což lze využít k implementaci herního prostředí.

3.3 UŽIVATELSKÉ ROZHRANÍ (OVLÁDÁNÍ PROGRAMU)

V druhém sešitě *Data* se nachází veškeré ovládání celého programu. Uživatel si zde může nastavit pozici hráče, orientaci kamery, textury a pozice krychlí, nebo také nastavení obrazovky a zároveň má možnost přehledu o různých statistikách, kdy a kolik výpočtů program u jednotlivých snímků provedl.

Tento celek se blíže věnuje jednotlivým částem uživatelského rozhraní. Veškeré hodnoty se nastavují pomocí změny textové hodnoty buňky a následným potvrzením (kliknutím na tlačítko *Apply*). (obr. 3.1)



obr. 3.1: uživatelské rozhraní programu v prostředí Excelu

3.3.1 Horní lišta

Horní lišta obsahuje nadpis, který informuje uživatele, že se nachází v sekci pro ovládání hry a statistiky. Vlevo se nachází tlačítko *Start*, které po kliknutí hráčem spustí funkci *Init*, která inicializuje celou hru. V pravé části zase tlačítko pro ukončení hry.

3.3.2 Player position

Zde si uživatel může nastavit současnou pozici hráče v souřadnicích x, y, z . Po provedení funkce pohybu zde program automaticky aktualizuje hodnoty aktuálních pozic hráče.

3.3.3 Camera orientation

Podobně jako v sekci pro pozici hráče, zde uživatel nastavuje úhly otočení kamery. Po provedení funkce pohybu program také aktualizuje aktuální hodnoty kamery.

3.3.4 Stats

V sekci *Stats* jsou zahrnuty statistiky, včetně celkového počtu krychlí v herním prostředí, počtu celkových stran (počet krychlí vynásobený 6) a počtu různých barevných jednotek textury (*Pixel* – počet stran vynásobený 64, což je počet pixelů v textuře). Dále tato sekce poskytuje informace o tom, kolik pozic stran, pixelů a buněk program vypočítal. Tato data lze využít jako indikátor úspěšnosti při optimalizaci programu.

3.3.5 Variables

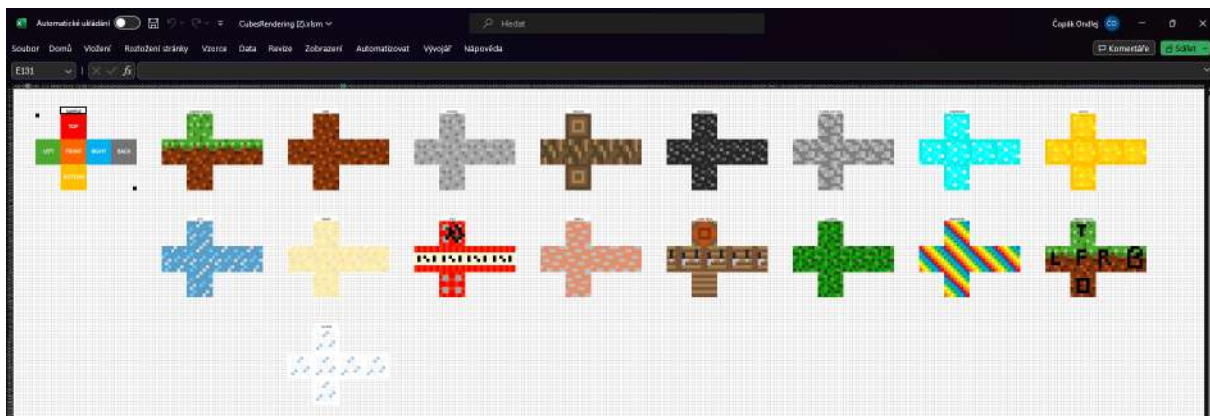
Zde může uživatel změnit výchozí hodnoty grafického výstupu. *Blocksize* určuje délku strany krychlí, a tím i jednoznačné měřítko souřadnic. *Screen Width* a *Screen Height* představují velikost obrazovky v počtu buněk. *Movement* a *Rotation Coefficient* určují měřítka pohybu hráče v konkrétních směrech a otočení kamery ve stupních podle konkrétních os. *InstantDrawing* ovlivňuje, zda se jednotlivé buňky budou vybarvovat postupně, což může sloužit ke grafické vizualizaci celého procesu zobrazování, který je zároveň časově náročnější. Zároveň se všechny buňky budou vybarvovat bez průběžného vykreslování programu, čímž se výrazně zrychlí zakreslování.

3.3.6 Blocks

Uživateli je zde nabídnuto měnit texturu a pozici středů jednotlivých krychlí. Výchozí hodnota textury je *NONE*, což znamená, že krychle nebude do programu začleněna. Pokud chce uživatel vložit krychli do programu, musí zvolit jednu z 16 předpřipravených textur (název) a následně zvolit její souřadnice, které by měly být vždy unikátní a násobky velikosti strany krychle (*Blocksize*), aby se krychle nepřekrývaly.

3.3.7 Texture List

Jedná se o seznam, který umožňuje uživateli vidět, jaké různé textury krychlí jsou k dispozici. Samotné textury jsou poté na 3. listu *Textures* (obr. 3.2).



obr. 3.2: seznam textur krychlí v 3. sešitu v Excelu

3.3.8 Timestamps

Poslední sekce slouží jako indikátor rychlosti programu. Každá buňka obsahuje přesný čas operace s přesností na milisekundu. V prvním řádku uživatel vidí čas, kdy byl program spuštěn, a následující řádky ukazují, kdy byly jednotlivé funkce pro výpočet snímku dokončeny. Poslední řádek udává celkový čas výpočtu snímku od provedení uživatelského vstupu po dokončení zobrazování buněk. Tento indikátor lze opět využít k optimalizaci a testování programu.

3.4 TŘÍDY (CLASSES)

Třídy usnadňují organizaci kódu a přispívají k modularitě a srozumitelnosti programu. Rozřazení dat a funkcí do logicky souvisejících celků usnadňuje správu a přehlednost programovacího prostředí, což je klíčové při vývoji komplexních systémů, jako je herní prostředí. Celkem program využívá 8 vlastních tříd.

3.4.1 Herní prostředí

3.4.1.1 Player

Třída `Player` slouží k uchování důležitých informací o hráči v herním prostředí. Tyto hodnoty zahrnují hráčovu aktuální pozici ve třech rozměrech (x, y, z) a úhly natočení kamery ($yaw, pitch$).

Využitím této třídy se zpřehlední kód, protože namísto používání 5 různých proměnných se používají proměnné jedné třídy. Navíc používání tříd má za výhodu lepší škálovatelnost programu, například při přidávání více hráčů.

3.4.1.2 Game

Funkce třídy `Game` je uchování všech různých nastavení scény a ovládání hry. Tato funkce obsahuje všechny nastavení týkající se grafického zpracování, která jsou uvedena v oddílu 3.3.5. Stejně jako u třídy `Player` slouží třída `Game` ke zpřehlednění programu. Zároveň by se tato třída dala využít k ukládání různých profilů nastavení hry nebo dalších nastavení prostředí.

3.4.1.3 Calculations

Třída `Calculations` optimalizuje výpočty goniometrických funkcí (cosinus, sinus, tangens) v herním prostředí. Obsahuje předem vypočtené hodnoty těchto funkcí pro úhly od 0 do 359 stupňů, což eliminuje nutnost opakovaných výpočtů během běhu programu. Použitím této třídy lze efektivněji zpracovávat úhly a minimalizovat tím zátěž spojenou s výpočty goniometrických funkcí při výpočtu pozic krychlí.

3.4.1.4 Textures

`Textures` je třída, která obsahuje funkce pro načítání a ukládání barevných textur. Metoda `LoadInput` načítá textury ze specifikovaného rozsahu buněk v listu a ukládá je do vnitřní kolekce (list) podle zadaného jména textury. Každá textura je reprezentována 8x8 maticí barev.

Metoda `GetColorCollection` umožňuje získání uložené kolekce barev asociované s určitou texturou, což zrychluje přístup k texturám pro vykreslování herních objektů.

3.4.1.5 Stats

Třída `Stats` slouží k ukládání podstatných statistik o herním prostředí v konkrétním snímku (*frame*). Uchovává informace o počtu krychlí (*Blocks*), viditelných pixelů (*VisiblePixels*), viditelných stran krychlí (*VisibleSides*), počtu vykreslených buněk (*Cells*) a vykreslených řádcích (*RowsDrawn*). Tyto data lze využít k vyhodnocení optimalizačních funkcí, například kolik stran krychlí nemusel program počítat, neboť jsou mimo zorné pole hráče. Stejně jako v případě tříd `Player` a `Game` slouží tato třída ke zpřehlednění kódu.

3.4.2 Geometrické objekty

3.4.2.1 Block

Třída **Block** uchovává informace o jednotlivých krychlech. Obsahuje proměnné, jako je *distance*, která představuje vzdálenost středu krychle od hráče, *point*, což je poloha středu krychle, a *sides*, kolekci reprezentující všech šesti stran krychle.

3.4.2.2 Side

Side je třídou, která podobně jako **Block** obsahuje informace o jednotlivých stranách krychle. Obsahuje data, jako je *distance*, což značí vzdálenost středu strany od hráče, *texture*, která určuje texturu strany (třída **Texture**), *middlePoint*, což je poloha středu strany, a *orientation*, udávající orientaci strany na (*Top/Bottom/Left/Right/Front/Back*). Dále obsahuje vrcholy strany označené jako *a*, *b*, *c* a *d*.

3.4.2.3 Pixel

Poslední třída **Pixel** je podobná třídě **Side**. Na rozdíl od této třídy však neobsahuje střed a vzdálenost od hráče. Místo toho obsahuje konkrétní barvu na základě pixelu textury. Každá instance třídy **Side** odpovídá vždy 64 instancím třídy **Pixel**.

3.5 PROCEDUREY (SUB, FUNCTIONS)

Celý program je rozdělen do pěti modulů (souborů), které tematicky sdružují funkce plnící různé úlohy při výpočtu snímku nebo interakci s prostředím Excelu. Podobně jako u tříd pomáhají moduly zpřehlednit kód.

V tomto prostředí se používají dva typy procedur: *Sub* a *Function*. Procedura *Sub* (*Sub-process* – podproces), na rozdíl od *Function* (Funkce), nedokáže vrátit hodnotu. Nicméně veškeré změny hodnot proměnných provedené v této proceduře zůstanou zachovány i po jejím ukončení. *Sub* procedury lze také přiřadit k tlačítkům a klávesovým zkratkám. *Function* se obvykle používají k provádění matematických operací a vždy musí vrátit předem deklarovanou hodnotu. *Sub* procedury se často využívají k interakci s Excelem, tj. práci s buňkami.

V této práci termín *funkce* zahrnuje obě procedury (funkce i podprocesy), se zaměřím na vybrané funkce, které jsou pro pochopení základního fungování programu nejpodstatnější. Detailněji se práce věnuje algoritmům pro optimalizaci v oddílu 4.3.

3.5.1 Main

Tento modul obsahuje nejdůležitější funkce pro vizualizaci herního prostředí. Hlavní část obsahuje procedury, které numericky zpracovávají vstupní hodnoty a vytvářejí z nich obraz.

3.5.1.1 Init (Sub)

Proces se spustí pouze po kliknutí na tlačítko. Po kliknutí proběhne načtení všech textur, pozic krychlí, pozic hráče, nastavení kamery a výpočtu hodnot goniometrických funkcí. Následně jsou zavolány funkce pro výpočet současného snímku a vytvoření klávesových zkratk.

3.5.1.2 Move (Sub)

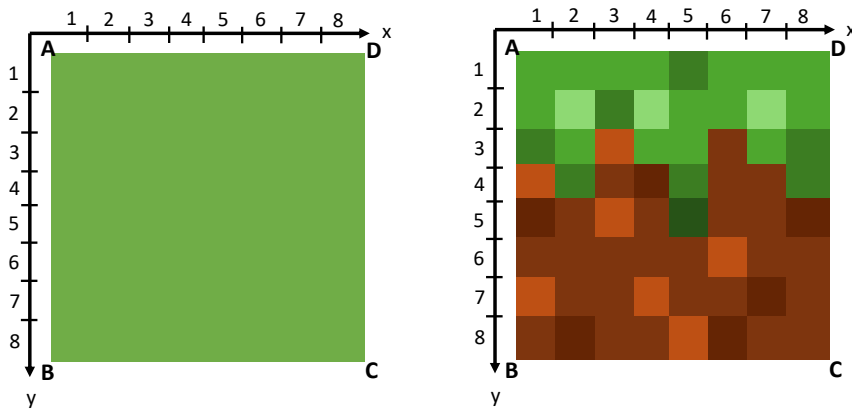
Nejprve se vymaže předchozí snímek tím, že se přebarví stejnou barvou. Poté se zde spustí veškeré funkce, které zajišťují zaznamenávání statistik a času spuštění programu. Dále jsou postupně zavolány tři funkce – `CalculatePositions`, `ApplyTexture` a `ConvertDraw2D`, na které se podrobně zaměřuji v kapitolách 3.5.1.3 až 3.5.1.5. Výsledkem těchto funkcí je kolekce všech pixelů, které vytvářejí výsledný snímek.

3.5.1.3 CalculateSides (Function)

Funkce nejdříve vytvoří kolekci tříd `Block` a přiřadí jim texturu a pozici na základě uživatelského vstupu. Poté založí novou kolekci tříd `Side` pro každou krychli s přiřazenými šesti stranami. Pro všechny vrcholy stran vypočítá nové pozice na základě polohy hráče a natočení kamery pomocí funkce `CalculateCoordinates`. Nakonec seřadí všechny strany podle vzdálenosti od hráče od nejvzdálenější po nejbližší, aby hráč nemohl vidět skrze strany.

3.5.1.4 ApplyTexture (Function)

Funkce `ApplyTexture` dostane na vstup kolekci `Side` a vrátí kolekci `Pixel`. Pro každou stranu funkce vytvoří 64 menších stran (objekt `Pixel`). Tyto pixely jsou definovány stejně jako strany – čtyřmi vrcholy a , b , c , d , které vznikají jako $\frac{k}{8}$ (pro k 1 až 8) násobkem vektorů \vec{ab} a \vec{ad} . (obr. 3.3), kód 3.5.1.4; řádek 5). Následně se aplikuje textura podle orientace (kód 3.5.1.4; řádek 16) a zkontroluje se, jestli se všechny vrcholy nacházejí v rovině před hráčem. Pokud ano, `Pixel` se uloží.



obr. 3.3: vizualizace aplikace textur – vlevo 1x Side, napravo 64x Pixel

```

1  For Each s In allSidesPre
2      For x = 0 To 7
3          For y = 0 To 7
4              ad = ((s.d(0) - s.a(0)) / 8, (s.d(1) - s.a(1)) / 8, (s.d(2) - s.a(2)) / 8)
5              ab = ((s.b(0) - s.a(0)) / 8, (s.b(1) - s.a(1)) / 8, (s.b(2) - s.a(2)) / 8)
6
7              a3 = (s.a + (ad(0) * x + ad(1) * x + ad(2) * x) + (ab(0) * y + ab(1) * y +
                  ab(2) * y))
8              b3 = (s.a + (ad(0) * x + ad(1) * x + ad(2) * x) + (ab(0) * (y + 1) + ab(1)
                  * (y + 1) + ab(2) * (y + 1)))
9              c3 = (s.a + (ad(0) * (x + 1) + ad(1) * (x + 1) + ad(2) * (x + 1)) + (ab(0)
                  * (y + 1) + ab(1) * (y + 1) + ab(2) * (y + 1)))
10             d3 = (s.a + (ad(0) * (x + 1) + ad(1) * (x + 1) + ad(2) * (x + 1)) + (ab(0)
                  * y + ab(1) * y + ab(2) * y))
11
12             If s.orientation = "top" Then
13                 col = textureColor(y + 1 + 18)(8 - x)
14             ElseIf s.orientation = "bottom" Then
15                 col = textureColor(y + 1)(8 - x)
16             Else
17                 col = textureColor(y + 1 + 9)(8 - x)
18             End If
19             If IsPointInsideFOV(a3, b3, c3, d3) = TRUE Then
20                 allSquares.Add Pixel.Initialize a3, b3, c3, d3, col
21             End If
22         Next y
23     Next x
24 Next s

```

kód 3.1: aplikace textury každé ze stran; upraveno

3.5.1.5 ConvertDraw2D (Function)

Poslední funkce `ConvertDraw2D` má za cíl převést všechny `Pixels` ze 3D do 2D a zobrazit je v konkrétních buňkách. Pro každý `Pixel` se vypočítají souřadnice podle algoritmu popsaného v oddíle 2.4. Předchozí část programu zabraňuje vstupu veškerých souřadnic, které mají souřadnici z zápornou.

Problém dělení 0, když souřadnice $z = 0$, se vyřeší nastavením na hodnotu $z = 1$, takže hráč bude tento objekt vidět. V poslední části cyklu se tyto body na škálují podle nastavení velikosti obrazovky (kód 3.5.1.5; řádek 8).

Následně se pro každý čtyřúhelník (`Pixel`) na základě všech 4 vrcholů zjistí veškeré souřadnice každé strany pomocí *Bresenhamova algoritmu* (viz oddíl 3.5.2.3). Funkce poté určí, které souřadnice na obrazovce leží uvnitř čtyřúhelníku, čímž získá všechny souřadnice každého `Pixelu`.

```
1  ' Converts all 4 vertexes from 3D to 2D
2  For Each intersectionPoint In Array(Side.a3, Side.b3, Side.c3, Side.d3)
3
4      ' Projects 3D points to 2D points
5      If Int(intersectionPoint(2)) = 0 Then
6          intersectionPoint = Array(intersectionPoint(0), intersectionPoint(1), 1)
7      End If
8
9      ' Applies Perspective Projection
10     projectedX = intersectionPoint(0) / intersectionPoint(2)
11     projectedY = intersectionPoint(1) / intersectionPoint(2)
12
13     ' Multiplies the projected points by the screen size
14     If G.screenHeight < G.screenWidth Then
15         screenX = Int((projectedX + 1) * 0.5 * G.screenHeight + (G.screenWidth - G.
             screenHeight) / 2)
16         screenY = Int((1 - projectedY) * 0.5 * G.screenHeight)
17     Else
18         screenX = Int((projectedX + 1) * 0.5 * G.screenWidth)
19         screenY = Int((1 - projectedY) * 0.5 * G.screenWidth + (G.screenHeight - G.
             screenWidth) / 2)
20     End If
21
22 Next intersectionPoint
```

kód 3.2: převod souřadnic ze 3D do 2D; upraveno

3.5.2 Geometry

V modulu `Geometry` se nacházejí všechny funkce, které se zabývají počítáním souřadnic.

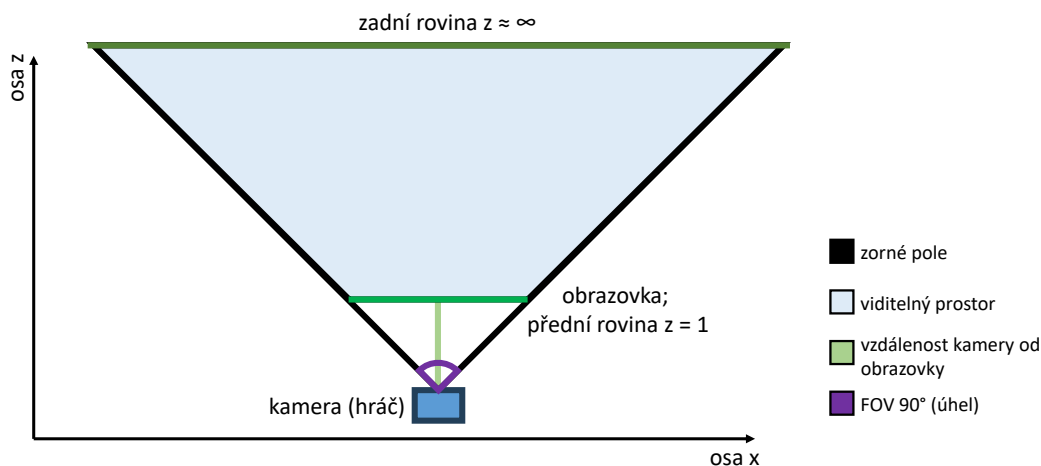
3.5.2.1 CalculateCoordinates (Function)

Tato funkce je implementací již zmíněného algoritmu v oddílu 2.1 a 2.2. Jako první se musí souřadnice jednotlivých stran posunout o opačné souřadnice pozice hráče a poté už nezáleží na pořadí otáčení kolem jednotlivých os X a Y . Výstupem funkce jsou nové souřadnice bodu v závislosti na parametrech hráče a kamery.

3.5.2.2 IsPointInsideFOV (Function)

Úkolem procedury jsou dvě věci. Ověřit, zda je daný bod v rovině před hráčem a také ověřit, zda bod leží v zorném poli hráče. První podmínka je splněna, pokud je $z > 0$.

Druhá podmínka vychází z matematického řešení převodu do 2D v oddílu 2.4. Zorné pole hráče připomíná komolý čtyřboký jehlan, kde horní podstava leží v rovině obrazovky hráče (rovina $z = 0$) a spodní podstava běží směrem k nekonečné souřadnici z (obr. 3.4) [5]. V každé rovině $z = k$, $k \in (0, \infty)$ je průnik dané k -té roviny se zorným polem obdélník, takže stačí spočítat, jestli se bod (na základě pozice z) nachází v něm. Nejprve je zapotřebí spočítat měřítko obdélníku. To také záleží na souřadnici z . Čím větší je souřadnice z , tím větší ten obdélník bude. Měřítko se vypočítá pomocí goniometrické funkce tangens (kód 3.5.2.2). Funkce zjistí délky menší strany a poté se spočítá délka druhé strany obdélníku v závislosti na poměru stran obrazovky (kód 3.5.2.2; řádek 2). Pokud jsou obě podmínky splněny, funkce vrátí hodnotu pravda.



obr. 3.4: zorné pole hráče, pohled shora – převedeno do 2D, pro lepší orientaci

```

1 radius = C.tan(90 / 2) * point(2)
2 If G.screenWidth > G.screenHeight Then
3     If (radius * G.screenWidth / G.screenHeight) - projectedPoint(0) >= 0 And radius -
        projectedPoint(1) >= 0 Then
4         result = TRUE
5     End If
6 Else
7     If radius-projectedPoint(0) >= 0 And (radius * G.screenHeight / G.screenWidth) -
        projectedPoint(1) >= 0 Then
8         result = TRUE
9     End If
10 End If

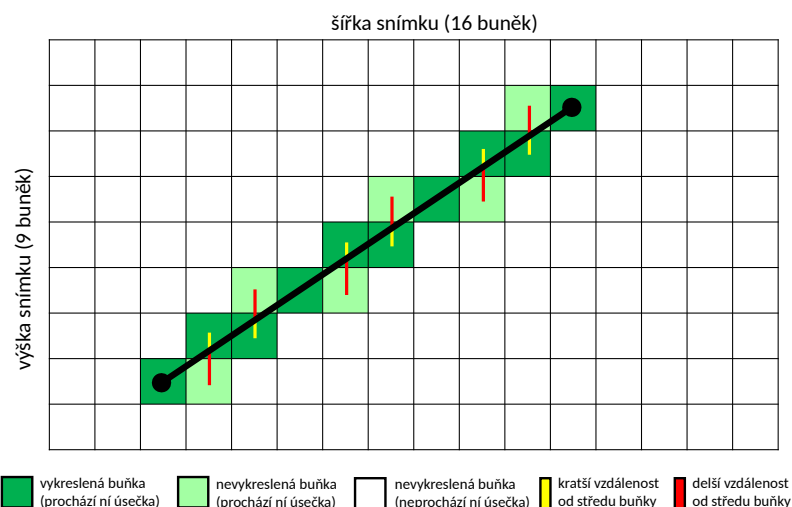
```

kód 3.3: kontrola, jestli se nachází bod uvnitř zorného pole; upraveno

3.5.2.3 GetLinePixels (Sub)

Závěrečná funkce modulu **Geometry** má za úkol vypočítat všechny 2D souřadnice úsečky mezi dvěma body. Dosáhnout výsledku lze za použití *Bresenhamova algoritmu*. Tento algoritmus využívá inkrementálního rozhodování o tom, který pixel vybrat pro vykreslení. Začíná výpočtem rozdílů mezi souřadnicemi koncových bodů, následně prochází body mezi nimi a vybírá z nich vhodné pixely (obr. 3.5) [6].

Dále se v rámci optimalizace programu ukládá do slovníku nejnižší a nejvyšší hodnota x podle klíče sloupce y . Tento proces se provádí pro každý bod čtyřúhelníku. Výsledkem je soubor pixelů ve formě čtverce, kde každé y obsahuje body od nejnižšího x do nejvyššího x . Toto se využívá při vykreslování buněk po řádcích v Excelu, kde nastavení barvy jedné buňky trvá stejně dlouho jako nastavení více buněk v obdélníkovém rozsahu [7].



obr. 3.5: Bresenhamův algoritmus pro získávání buněk na základě přímky

3.5.3 Functions

Zde se nacházejí funkce, které srovnávají kolekce (převážně strany krychlí), podle daných kritérií.

3.5.3.1 RemoveDuplicateSides (Sub)

Tato optimalizační funkce odstraní veškeré strany, které mají stejný střed. Pokud jsou všechny krychle ve vzdálenosti násobků délek stran, platí, že každý střed může mít pouze strany se stejnou orientací. Zároveň, pokud se v jednom místě nenachází více krychlí, tak v každém středu strany se mohou nacházet maximálně dvě strany.

Funkce dosahuje výsledku tím, že převede kolekci (list) na knihovnu indexovanou podle středu a následně zpětně na kolekci. Vždy když program najde střed, který již existuje tak se uloží do nového listu a strany s těmito středy jsou vyřazeny.

3.5.3.2 ReverseCollection (Function)

Jedná se o jednu ze základních funkcí, kterou bylo nutné manuálně implementovat. Tato funkce vrátí nový list, který je seřazen od posledního prvku po první.

3.5.3.3 SortByDistance (Function)

Výstupem této funkce je seřazený list sestupně podle vzdálenosti stran od hráče. Procedura využívá vlastní implementaci řazení čísel pomocí *QuickSortu*.

3.5.3.4 QuickSort (Sub)

QuickSort postupně rozděljuje list (kolekce) čísel na menší listy pomocí vybraného prvku zvaného *pivot*. Prvky jsou poté uspořádány tak, že prvky menší než *pivot* jsou nalevo a větší jsou napravo [8]. Tento proces je opakován rekurzivně, dokud není list seřazen.

3.5.4 Visual

Předposlední soubor obsahuje funkce, které komunikují s Excelem. Funkce se týkají především vybarvováním buněk, zapisováním hodnot a načítáním nastavení.

3.5.4.1 ClearScreen (Sub)

Úkolem této procedury je vyčistit poslední snímek od všech zobrazených krychlí. Funkce toho dosáhne tím, že nastaví barvu všech buněk na jednolitou barvu, defaultně modrou, která představuje nebe.

3.5.4.2 FillCellsInRange (Sub)

Tato funkce dostane na vstupu souřadnice levého horního rohu, pravého spodního rohu a barvu pro daný rozsah buněk. Následovně tato funkce ověří, že všechny buňky se nacházejí na obrazovce, a ty vybarví zvolenou barvou (na základě textur).

3.5.4.3 SetPlayer/Variables (Sub)

Obě funkce načtou informace ohledně pozic hráče/nastavení herního prostředí z 2. listu (nastavení) do paměti programu. Obě funkce se volají vždy před začátkem výpočtu současného snímku.

3.5.5 Keys (Ovládání)

Poslední modul se stará o kontrolování hry pomocí klávesových zkratk. Ovládání hry pomocí klávesnice je důležitým prvkem k zjednodušení ovládání hry, místo klikání na tlačítka (na obrazovce) nebo manuálního nastavování číselných pozic hráče.

Excel neumožňuje přiřazovat jednotlivým klávesám možnost spuštění funkcí, proto ke všem klávesovým zkratkám byla přiřazena klávesa *levý shift*. Proto namísto pohybu dopředu pomocí klávesnice *w* se hra ovládá pomocí *shift+w*. V následujících funkcích je shift z klávesových zkratk z důvodu přehlednosti vypuštěn.

Po spuštění každé z funkcí se nejprve upraví pozice hráče nebo natočení kamery vzhledem ke klávesové zkratce a vzápětí se spustí funkce pro výpočet současného snímku. Proměnné *moveBy* a *rotateBy* určují hodnotu, o kolik se hráč pohybuje a otáčí a lze je měnit v sešitu nastavení. V práci symbol *+-* označuje, zda je hodnota kladná nebo záporná v závislosti na směru pohybu hráče.

3.5.5.1 BindKeys (Sub)

Po kliknutí na tlačítko *Start* se spustí tato funkce. Nastaví veškeré klávesové zkratky sloužící k ovládání hry z procedur, které jsou níže uvedeny.

3.5.5.2 FreeKeys (Sub)

Na rozdíl od BindKeys tato funkce naopak vymaže všechny klávesové zkratky sloužící k ovládání hry.

3.5.5.3 MoveTop/Bottom (Sub)

Klávesové zkratky: *mezerník/x*

Pohyb po ose Y

Na rozdíl od pohybu po osách X nebo Z , nezáleží na otočení kamery, proto při pohybu hráče nahoru a dolu se mění pouze po ose Y .

```
1 P.y = P.y + G.moveBy ' for Top
2 P.y = P.y - G.moveBy ' for Bottom
```

kód 3.4: pohybování hráče nahoru, dolu; upraveno

3.5.5.4 MoveLeft/Right (Sub)

Klávesové zkratky: *a/d*

Pohyb po osách X, Z

Zde je potřeba vzít v úvahu otočení kamery hráče podle osy Y . Pokud by se hráč nacházel v bodě $[0, 0, 10]$, otočení kamery y by bylo 0 , tak hráč by se měl pohybovat pouze po ose X . V případě, že by hráč stál v bodě $[10, 0, 0]$ a kamera by byla $y = -90^\circ$, tak se hráč musí pohybovat po ose Z . Proto je nutné použít goniometrické funkce v závislosti na otočení kamery.

```
1 dx = C.sin(P.yaw) * G.moveBy
2 dz = C.cos(P.yaw) * G.moveBy
3
4 ' for Left -; +
5 ' for Right +; -
6 P.x = P.x +- dz
7 P.z = P.z +- dx
```

kód 3.5: pohybování hráče doleva, doprava; upraveno

3.5.5.5 MoveFront/Back (Sub)

Klávesové zkratky: *w/s*

Pohyb po osách *X, Z*

Stejně jako u pohybu doleva a doprava, i při pohybu nahoru a dolů se opět pracuje se souřadnicemi vztaženými k hráči, nikoliv k osám. Proto je také nutné stejně zohlednit otočení kamery podle osy *Y* pomocí stejné úvahy.

```
1 dx = C.sin(P.yaw) * G.moveBy
2 dz = C.cos(P.yaw) * G.moveBy
3
4 ' for Front +; +
5 ' for Back -; -
6 P.x = P.x +- dx
7 P.z = P.z +- dz
```

kód 3.6: pohybování hráče dopředu, dozadu; upraveno

3.5.5.6 LookTop/Bottom (Sub)

Klávesové zkratky: *r/f*

Otočení kamery podle osy *X*

Hodnota otáčení podle osy *X* (*pitch*) v rozmezí $(-90; 90)$ stupňů vyjadřuje úhel natočení hlavy hráče a omezuje možnost pohledu směrem nahoru a dolů. Tato omezení odpovídají reálným fyzickým schopnostem pohybu hlavy člověka.

```
1 P.pitch = P.pitch +- G.rotateBy
2
3 ' for Top >; +; -
4 ' for Bottom <; -; +
5 If P.pitch >(<) +-90 Then
6     P.pitch = +-90
7 End If
```

kód 3.7: otáčení kamery nahoru, dolů; upraveno

3.5.5.7 LookLeft/Right (Sub)

Klávesové zkratky: q/e

Otočení kamery podle osy Y

V případě otáčení (*celého těla hráče*) podle osy Y (*yaw*) v rozmezí $(0; 360)$, které je udáváno ve stupních, umožňuje program hráči otáčet se kolem vlastní osy. Pokud hráč překročí horní hranici tohoto rozsahu, hodnota otáčení se sníží nebo zvýší o jednu periodu (360°). Tímto způsobem se zajistí, že program může využívat předem spočtených hodnot goniometrických funkcí, což usnadňuje výpočty v rámci hry.

```
1 P.yaw = P.yaw +- G.rotateBy
2
3 ' for Left >=; 360; +
4 ' for Right <=; 0; -
5 If P.yaw >(<=) 360(0) Then
6     P.yaw = P.yaw +- 360
7 End If
```

kód 3.8: otáčení kamery doleva, doprava; upraveno

3.6 MOŽNOSTI KRYCHLÍ

Program pracuje s čtyřúhelníky jako základními prvky. Všechny tvary, především krychle, jsou vytvářeny na základě čtyř vrcholů, které jsou analyticky vyjádřeny.

3.6.1 Krychle

Krychle je základním objektem, ke kterému byl program převážně vytvořen. Každá krychle má 6 stran a každá z nich má rozměry 8x8 Pixelů.

3.6.2 Půlkrychle

Namísto krychle je výsledným tělesem kvádr, který má přesně poloviční výšku. Na tento objekt lze aplikovat normální texturu (stejnou jako pro krychli). Horní a spodní stěny zůstávají stejné jako u krychle, a na bočních stěnách se použijí spodní části textury o rozměrech 8x4.

3.6.3 Přidávání jiných tvarů

Program je vytvořen tak, aby bylo pro uživatele jednoduché přidávat vlastní tělesa. Nejprve je zapotřebí do funkce `InitBlock (Main)` vložit konstrukční metody, které definují středy stěn a jejich orientaci vůči středu tělesa, aby bylo možné později aplikovat textury. Dále v inicializační funkci (`Side`) uživatel musí určit pozici vrcholů čtyřúhelníku vůči danému bodu. Zde je například možné namísto čtverců vytvořit trojúhelníky, ze kterých se v běžných 3D enginech skládají veškeré objekty, pokud uživatel nastaví stejné souřadnice pro dva různé body (a tím zredukuje počet vrcholů ze čtyř na tři).

Problém v případě použití trojúhelníků by byla následovná aplikace textur, jelikož u spojeného vrcholu by textura byla více intenzivní než na opačné straně. Tento problém by byl potřeba vyřešit v poslední části sloužící k přidání nových tvarů (funkce `ApplyTexture` v `Main`). Zde uživatel určuje, jak se bude daná textura aplikovat (a jaké části).

3.6.4 Průhlednost

V Microsoft Excelu není možné nastavit průhlednost barvy výplně buněk. Uživatel namísto toho může do buněk textur vkládat desetinné číslo od 0 do 1, které reprezentuje intenzitu *neprůsvitnosti* barvy v textuře (0 = zcela průhledné, 1 = zcela neprůhledné). Detekce čísel probíhá ve třídě `Textures (LoadInput)`. V případě, že buňka neobsahuje žádný text, je *neprůhlednost* nastavena na 1.

Během běžného vykreslování se jednotlivé buňky překreslují. V případě průhledné textury se nejprve zjistí, jaká barva se v buňce nachází, a vykreslí se nová barva v poměru s průhledností nové barvy. Tento proces není slučitelný s vykreslováním po řádcích, protože nově vytvořené barvy nemusí být stejné (pokud barvy buněk před vykreslením průhledné textury budou různé), a proto je nutné vykreslit průhledné textury vždy buňku po buňce (kód 3.6.4).

```
1  ' Get the current RGB color
2  originalCol = originalCell.Interior.Col
3  ' Calculate the new color
4  newColor = RGB(
5      Int(((opacity) * (originalCol % 256) + opacity * (colTexture % 256)) + 0.5),
6      Int(((opacity) * ((originalCol \ 256) % 256) + opacity * ((colTexture \ 256) %
7          256)) + 0.5),
8      Int(((opacity) * (originalCol \ 65536) + opacity * (colorTexture \ 65536)) + 0.5)
9  )
10 originalCell.Interior.Color = newCol
```

kód 3.9: aplikace průhledné buňky; upraveno

4 OPTIMALIZACE PROGRAMU

Poslední část práce se zabývá analyzováním programu a hledáním možností optimalizací.

4.1 LIMITACE VBA

Při práci ve 3D prostředí není VBA optimálním jazykem. Na rozdíl od počítačové hry Minecraft není prostředí Excelu koncipováno pro zpracování velkého množství výpočtů v krátkém čase. VBA je především navržen pro automatizaci v aplikacích od Microsoftu [9]. Jazyk není schopen provádět více částí v kódu současně (tzv. *vícevláknovost*) a zároveň nemá možnost využívat *GPU*, grafický procesor určený pro zpracování velkých objemů dat [9].

Dalším problémem je využívání statických typových kontrol (proměnné nejsou vázány na konkrétní datový typ, ale mohou být v průběhu programu měněny). To výrazně zpomaluje kód kvůli nutnosti dynamické adaptace k různým datovým typům [9].

4.2 SLOŽITOSTI FUNKCÍ

Tato část se věnuje tzv. *Landauovi notaci* (v angličtině *Big O Notation*), která určuje náročnost výpočtu v závislosti na růstu parametru. Při hodnocení složitosti každé funkce se vždy upřednostňuje notace nejvyššího řádu [10]. To platí zejména v případech, kdy funkce kombinuje více podfunkcí, které nejsou vzájemně závislé v rámci cyklů. Složitost se v této práci značí pomocí $O(f(x))$.

4.2.1 Init

Jedná se o lineární notaci $O(n)$, která závisí na počtu textur nebo počtu hodnot gonio-metrických funkcí.

4.2.2 CalculatePosition

Funkce se chová jako $O(n)$, kde n je počet krychlí. Pro každou krychli se spočítá 30 pozic (6 stran, 4 vrcholy a střed) a následně se strany seřadí podle vzdálenosti pomocí *QuickSortu* (oddíl 3.5.3.4), což je v průměru $O(n \log n)$ [8].

4.2.3 ApplyTexture

Tato funkce také vykazuje lineární složitost $O(n)$. Prochází všechny strany a rozděljuje je na 64 částí. Koeficient bývá poměrně vysoký, v řádech desítek, kvůli všem početním funkcím, avšak zůstává konstantní v závislosti na počtu stran.

4.2.4 ConvertDraw2D

Tato funkce má největší množství výpočtů. Pro každou stranu se vypočítají 2D souřadnice všech čtyř vrcholů. Program poté musí určit buňky, které nastavit na danou barvu na základě pozice všech bodů ležících ve čtyřúhelníku, což se děje pomocí *Bresenhamova algoritmu* (oddíl 3.5.2.3).

Celková složitost této funkce je obtížná k určení, a může dosahovat až $O(n \cdot p^3)$, kde n je počet stran a p je průměrná délka strany (v buňkách), která v blízkosti hráče může dosahovat hodnot blížících se nekonečnu.

4.2.5 Celková

Vzhledem k tomu, že tyto funkce jsou vykonávány postupně a každá pouze jednou, celková komplexita se řídí notací funkce *ConvertDraw2D*, která má nejvyšší časovou náročnost.

Ostatní funkce mají časovou náročnost převážně kolem $O(n)$. Při zpracování velkého množství souřadnic, což platí již při výpočtu jedné krychle (ve vzdálenosti od hráče 1 až 30 bloků s náhodným natočením vůči kameře), jsou tyto funkce z hlediska časové náročnosti zanedbatelné ve srovnání se složitostí třetího stupně.

4.3 IMPLEMENTACE OPTIMALIZACÍ

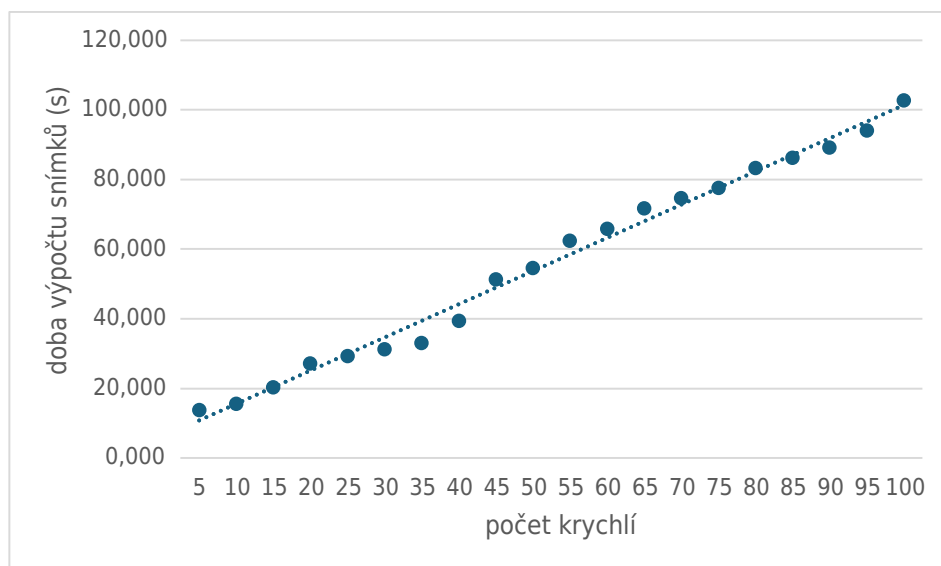
Pro plynulý běh by počítačová hra měla být schopna vykreslit jeden snímek za $\frac{1}{60}$ sekundy (*fps*). Cílem je zredukovat počet výpočtů, aby došlo k zrychlení. Toho lze ve VBA dosáhnout pomocí hledání efektivnějších algoritmů a vynecháním objektů, které hráč nevidí.

4.3.1 Závislost výkonu

V prvním segmentu optimalizace se detailněji zaměřím na to, jak růst některých z parametrů může ovlivnit celkovou rychlost programu. Tato analýza se opírá o naměřené hodnoty z provedených testů, které byly realizovány na celkově optimalizovaném kódu. Konkrétně se jedná o program, který integruje všechny optimalizace (oddíl 4.3.4). Dále se zaměřím na vztah mezi růstem počtu krychlí a počtu vykreslovaných buněk s výkonem programu.

4.3.1.1 Počet krychlí

Test zkoumal závislost času vykreslování snímku na rostoucím počtu krychlí. Testy byly provedeny na sadě od 5 do 100 krychlí s krokem 5 krychlí. Krychle byly náhodně rozmístěny v blízkém okolí hráče. Z grafu (obr. 4.1) vyplývá, že závislost je lineární, a s rostoucím počtem krychlí se doba vykreslování na jednu krychli blíží jedné sekundě. To znamená, že vykreslení jedné krychle trvá programu zhruba 1 sekundu (tab. 4.1).



obr. 4.1: závislost doby výpočtu jednoho snímku na celkovém počtu vykreslovaných krychlí (optimalizovaný kód)

Počet krychlí	5	10	15	20	25	30	35	40	45	50
Doba výpočtu snímku [s]	13,715	15,616	20,354	27,087	29,257	31,233	33,010	39,448	51,343	54,532
Čas / krychle [s]	2,743	1,562	1,357	1,354	1,170	1,041	0,943	0,986	1,141	1,091

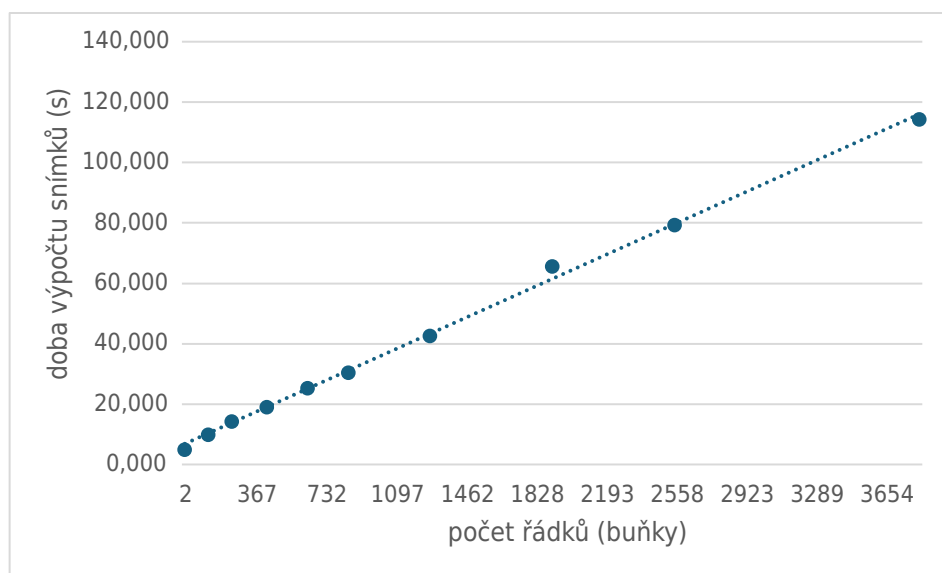
Počet krychlí	55	60	65	70	75	80	85	90	95	100
Doba výpočtu snímku [s]	62,370	65,750	71,733	74,558	77,524	83,243	86,227	89,180	94,052	102,588
Čas / krychle [s]	1,134	1,096	1,104	1,065	1,034	1,041	1,014	0,991	0,990	1,026

tab. 4.1: doba výpočtu jednoho snímku v závislosti na celkovém počtu krychlí (optimalizovaný kód)

4.3.1.2 Počet buněk

Test zkoumal závislost času vykreslování snímku na rostoucím počtu buněk. Testy byly provedeny v rozpětí od výšky řádku 9 do 2160 buněk v posloupnosti běžně používaných hodnot od $nHD(360)$ do $4K UHD(2160)$ s poměrem obrazovky 16 : 9 [11].

Z výsledků lze usoudit, že s rostoucím počtem buněk výrazně klesá čas na výpočet jedné buňky. Toto pravidlo platí, i když se čas vztáhne na počet řádků (*buněk*) (obr. 4.2). To znamená, že níže optimalizovaný kód skutečně urychluje vykreslování buněk po řádcích. Z testů vypadá, že limitní hodnota pro řádek bude blízká 0,03 sekundy na řádek (tab. 4.2).



obr. 4.2: závislost doby výpočtu jednoho snímku na počtu vykreslovaných buněk (optimalizovaný kód)

Celkový počet buněk	144	9216	36864	102400	230400	409600	921600	2073600	3686400	8294400
Šířka obrazu [buněk]	16	128	256	427	640	853	1280	1920	2560	3840
Výška obraz [buněk]	9	72	144	240	360	480	720	1080	1440	2160
Doba výpočtu snímku [s]	5,031	9,804	14,196	19,055	25,375	30,391	42,555	65,640	79,297	114,359
Čas / 1000 buněk [s]	34,938	1,064	0,385	0,186	0,110	0,074	0,046	0,032	0,022	0,014
Čas / počet řádků [s]	0,314	0,077	0,055	0,045	0,040	0,036	0,033	0,034	0,031	0,030

tab. 4.2: doba výpočtu jednoho snímku v závislosti na celkovém počtu vykreslovaných buněk (optimalizovaný kód)

4.3.2 Sady testů

Všechny optimalizace byly testovány na základě 5 sad testů, které vycházejí z 10 různých měření. Každá sada testů byla navržena s ohledem na časovou povahu a využívala stejné náhodně umístěné krychle napříč měření. Průměrná časová zlepšení pro každou optimalizaci jsou zaznamenána v tab. 4.3.

Výsledky měření se mohou lišit v závislosti na zařízení, na kterém byly provedeny. Měření doby vykreslování stejných snímků se liší například kvůli prioritizaci procesoru, který může v daný moment prioritizovat jiné úlohy, množství dostupné paměti *RAM* a další [12]. Pokud program z časových důvodů nedokázal vykreslit snímek, je v tabulce místo něj uvedena *pomlčka*. Naměřené hodnoty jsou uvedené v příloze práce a program tyto hodnoty pročítal přes 16 hodin.

- Hráč ve středu jedné krychle, náhodné otáčení kamery.
- 50 náhodně umístěných krychlí v blízkosti hráče. (Nemusí být v zorném poli hráče, nezapočítává se do průměru)
- 50 náhodně umístěných krychlí v blízkosti hráče bez zakreslování pixelů.
- 50 náhodně umístěných krychlí v blízkosti hráče bez všech výpočtů, pouze zakreslování pixelů. (Určeno jako rozdíl předchozích dvou měření)
- Pohled na 27 sousedících krychlí s konstantními parametry pozic a otáčení.

4.3.3 Bez optimalizace

Bez jakýchkoliv optimalizací měl program problém s některými výpočty stran v blízkosti hráče, a dokonce program nedokázal tyto hodnoty spočítat kvůli chybě přetečení hodnot (*overflow*) u některých kolekcí. Výpočet 50 krychlí trval programu v průměru kolem **20 minut**, a průměrný čas všech testů se pohyboval kolem **8 minut** (tab. 4.3). Tento čas je výrazně dlouhý a je nutné ho zkrátit.

4.3.4 Konkrétní optimalizace

4.3.4.1 Zorné pole hráče

Tato optimalizace výrazně snižuje počet výpočtů pro všechny body, které hráč na obrazovce nemůže vidět, vycházející z jeho zorného pole, které je v tomto programu 90°. To vede ke snížení výpočtů pro množství 2D souřadnic v blízkosti hráče. Největší zlepšení výkonu je patrné v *1. sadě testů*, kde se program v průměru zlepšil o **68 %** (tab. 4.4). Podobné zlepšení je také viditelné při testu s *náhodně umístěnými 50 krychli*, zejména když se hráč nachází ve více krychlích současně (tab. 4.4).

Tato optimalizace nemá výrazný vliv při testu s *27 sousedícími krychlemi*, které jsou již v zorném poli hráče. Stejně tak není zlepšení patrné při testu s *50 krychlemi bez zakreslování*, protože optimalizace se nevztahuje k samotnému kreslení barev buněk do sešitu (tab. 4.4). Implementace algoritmu je v oddíle 3.5.2.2.

4.3.4.2 Počet stran krychle

Jedná se o algoritmus, který snižuje počet zobrazených stran pro každou krychli. Pokud se hráč dívá na krychli a není uvnitř, ale vně krychle, může v jednom okamžiku vidět maximálně tři různé strany (z jednoho bodu). Při srovnání původních dat, kde byly testované všechny optimalizace, s daty, kde byla tato optimalizace vynechána, lze pozorovat **40 %** zrychlení při testu *zobrazování 27 krychlí* (tab. 4.5). Ve srovnání s programem bez jakýchkoliv optimalizací je však toto zrychlení pouze **8 %** (tab. 4.4). *Uvnitř krychle* je dokonce rychlejší než při použití celkové optimalizace (tab. 4.5), což je způsobeno odchylkou zmíněnou výše.

4.3.4.3 Duplicitní strany

Tato optimalizace vyřazuje strany, které se nacházejí blízko sebe. Funkce vyřazuje obě strany, protože v případě, kdy se krychle dotýkají, ani jedna z dotýkaných stran nemůže být vidět (za předpokladu, že jsou krychle umístěny v násobcích svých stran). Nejvýraznější zlepšení se projevuje v testu, kde se všech *27 krychlí* dotýká. V porovnání s programem bez optimalizace se kód v průměru zrychlil o **12 %** (tab. 4.4) a oproti celkové optimalizaci o **60 %** (tab. 4.5).

4.3.4.4 Vykreslování po řádcích

Vykreslování po řádcích je optimalizace, kde program vybarvuje buňky po řádcích podle současné barvy textury místo toho, aby vybarvoval každou buňku zvlášť. Vybarvit jednu buňku trvá stejně dlouho jako vybarvit několik buněk (obdélník buněk) stejnou barvou. Proto tato optimalizace zlepšuje výkon, zejména v sadě testů *50 krychlí se zakreslováním*, a to o **95 %** v porovnání s celkovou optimalizací (tab. 4.4) a **25 %** vůči programu bez optimalizací (tab. 4.4). Navíc dochází ke snížení složitosti zakreslování buněk do sešitu z $O(n^2)$ na $O(n)$.

4.3.5 Veškeré optimalizace

Kombinací všech pěti uvedených optimalizací dojde k výraznému zvýšení efektivity programu a zkrácení času vykreslování v průměru o **96 %** (tab. 4.4). Průměrný snímek v dané sadě testů trvá **15 sekund** místo původních **491 sekund** (tab. 4.3). I přes tyto zlepšení zůstává nejnáročnějším procesem výpočet konkrétních buněk na základě vrcholů čtyřúhelníků (oddíl 4.2.4).

4.3.6 Další možnosti

Další možnosti, jak zefektivnit program, zahrnují optimalizaci algoritmu pro výpočet pozic buněk u lichoběžníků. Současný algoritmus musí procházet všechny buňky mezi všemi vrcholy, což může být neefektivní, zejména při malých vzdálenostech s vysokými hodnotami.

Výraznou optimalizací by mohlo být omezení výpočtu pozic krychlí a stran, které nejsou viditelné hráčem za ostatními krychlemi. Také by se mohlo zvážit přestat zobrazovat textury krychlí, ve kterých se hráč nachází. Celkově lze program dále optimalizovat, avšak kvůli povaze prostředí nelze dosáhnout vyšší vykreslovací rychlosti blízké $\frac{1}{60}$ sekundy.

průměrné hodnoty [čas / snímek]	Uvnitř jedné krychle	27 sousedících krychlí	50 náhodných krychlí	50 krych. bez zakreslní	50 krych. pouze zakres.	Průměr
Bez optimalizací [s]	83,581	462,391	1420,584	1144,963	275,621	491,639
Zorné pole hráče [s]	26,709	439,961	701,132	429,356	271,776	291,951
Počet stran krychle [s]	78,337	422,852	1369,481	1099,550	269,931	467,667
Duplicitní strany [s]	78,013	404,532	1384,568	1114,410	270,158	466,778
Vykreslování po řádcích [s]	49,766	427,275	1322,370	1117,569	204,801	449,853
Celková Optimalizace [s]	5,437	22,185	31,125	28,199	2,927	14,687

tab. 4.3: doba trvání průměrného vykreslení obrázku v sekundách

průměrné hodnoty [čas / snímek]	Uvnitř jedné krychle	27 sousedících krychlí	50 náhodných krychlí	50 krych. bez zakreslní	50 krych. pouze zakres.	Průměr
Bez optimalizací [%]	0,000	0,000	0,000	0,000	0,000	0,000
Zorné pole hráče [%]	68,044	4,851	50,645	62,500	1,395	34,197
Počet stran krychle [%]	6,275	8,551	3,597	3,966	2,065	5,214
Duplicitní strany [%]	6,663	12,513	2,535	2,668	1,982	5,957
Vykreslování po řádcích [%]	40,458	7,594	6,914	2,393	25,695	19,035
Celková Optimalizace [%]	93,495	95,202	97,809	97,537	98,938	96,293

tab. 4.4: poměr optimalizací vůči programu bez žádných optimalizací vztaženo k bez optimalizací v procentech

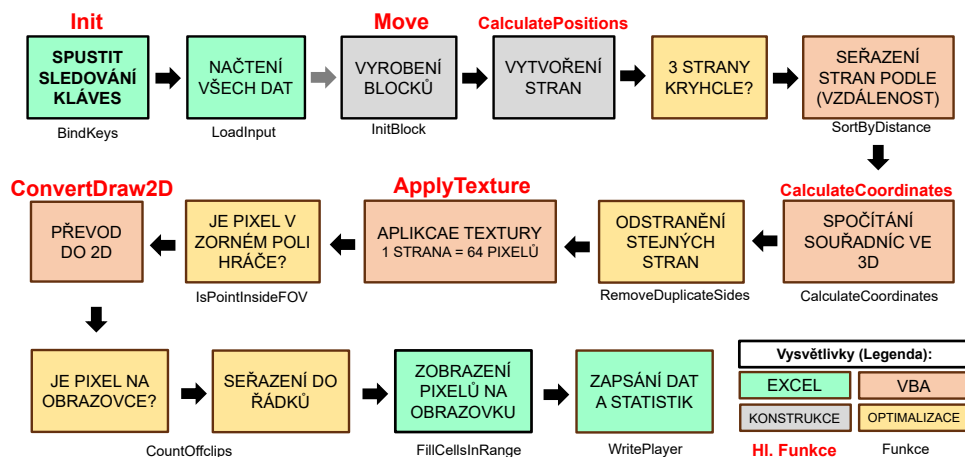
průměrné hodnoty [čas / snímek]	Uvnitř jedné krychle	27 sousedících krychlí	50 náhodných krychlí	50 krych. bez zakreslní	50 krych. pouze zakres.	Průměr
Bez optimalizací [%]	0,000	0,000	0,000	0,000	0,000	0,000
Zorné pole hráče [%]	90,440	1,091	95,674	96,059	23,886	52,869
Počet stran krychle [%]	-3,663	43,891	39,093	37,906	48,569	31,676
Duplicitní strany [%]	2,369	61,657	13,579	7,704	46,433	29,541
Vykreslování po řádcích [%]	83,921	36,824	68,309	-2,938	95,868	53,419
Celková Optimalizace [%]	93,495	95,202	97,809	97,537	98,938	33,501

tab. 4.5: poměr celkové optimalizace vůči konkrétní optimalizaci vztaženo k celkové optimalizaci v procentech

4.4 SHRNUŤÍ STRUKTURY PROGRAMU

Počítání snímku se dá spustit pomocí **Init**, které následovně spustí **Move**, nebo pomocí kláves, sloužících k pohybu hráče, které také spustí **Move**. Po načtení všech proměnných a parametrů prostředí se začnou vyrábět datové struktury **Block** a **Side**. Následně se zobrazí vždy 3 nejbližší strany každé z krychlí (pokud je hráč mimo krychli) a všechny strany se seřadí podle vzdálenosti od nejvzdálenější po nejbližší.

Dále se spočítají veškeré souřadnice vrcholů a středy každé ze stran. Odstraní se strany, jenž sdílí stejný střed. Každá ze stran se rozdělí na 64 menších stran, které už mají konkrétní barvu na základě textury. Zkontroluje se, zdali se nachází v jeho zorném poli a tyto stejnobarevné části se převedou do 2D na konkrétní buňky, ty se převedou do příkazů (vybarvení řádků) a zkontroluje se, aby všechny buňky ležely v obrazovce. Po zobrazení se zapíší do programu statistiky a program vyčkává na další vstup. (obr. 4.3)(obr. 4.4)



obr. 4.3: graf základní struktury celého programu



obr. 4.4: příklad vykreslení snímku pomocí programu

5 DISKUSE

Tato předposlední část práce se zabývá možnostmi zlepšení programu, způsobem, jak na tuto práci navázat a aplikovat tento program.

5.1 OPTIMALIZACE

Největší prostor na zlepšení programu se týká jeho rychlosti. I přes implementované optimalizace nedosahuje program požadované standardní vykreslovací rychlosti blízké $\frac{1}{60}$ sekundy (60 snímků za sekundu). Existují další metody zrychlení, ale vyžadují externí programy mimo Excel. Kvůli cíli práce, kterým je renderování (vykreslování) pouze pomocí VBA v Excelu, jsou tyto možnosti uvedeny stranou a mohou posloužit pro budoucí vývoj.

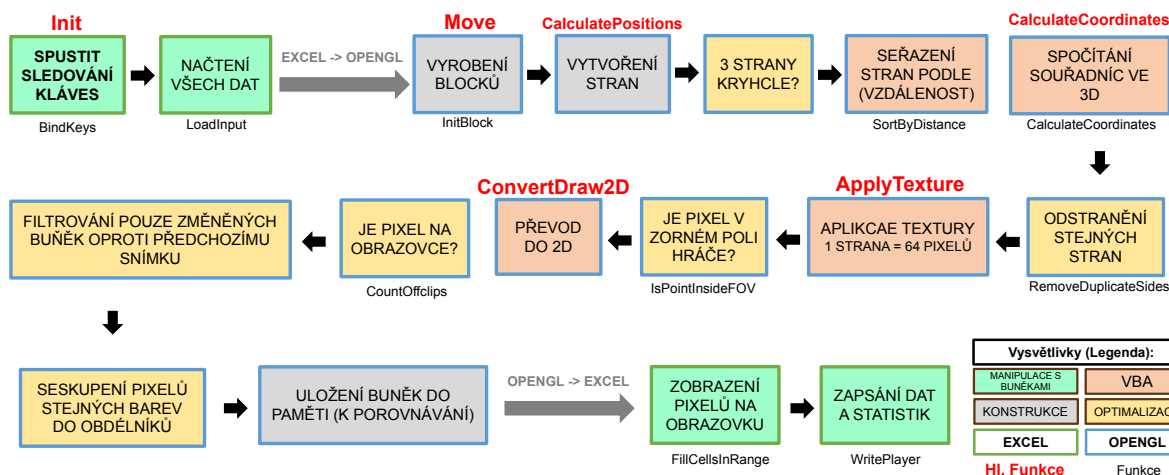
5.1.1 GPU

Hlavním omezením jazyka VBA je nemožnost využití GPU [9]. To by umožnilo zrychlit výpočty souřadnic, jelikož je k tomu GPU přímo určeno. Existují externí knihovny umožňující komunikaci mezi VBA a GPU pomocí rozhraní jako *OpenCL*¹, optimalizované pro práci s 3D vykreslováním².

Konkrétní implementace by mohla fungovat následovně: Nejprve by VBA v Excelu shromáždilo veškerá data z jednotlivých buněk, které by prostřednictvím textového souboru nebo jako parametry předalo (zatím neimplementovanému) programu v jazyce C využívajícího *OpenCL*. Tento program by následně provedl veškeré výpočty, analogicky k tomu, co je již naprogramováno ve VBA. Poté by vrátil výsledky buď formou návratové hodnoty, nebo pomocí textového souboru obsahujícího seznam jednotlivých buněk k vybarvení zpět do původního programu. Tato implementace by umožnila snížení času veškerých početních operací na milisekundy (pro jeden snímek)(obr. 5.1).

¹OpenCL (1992, Khronos Group)

²Externí knihovna (2023, github.com/Excel-lent/ClooWrapperVBA, Excel-lent)

obr. 5.1: graf základní struktury programu s výpočty v *OpenCL*

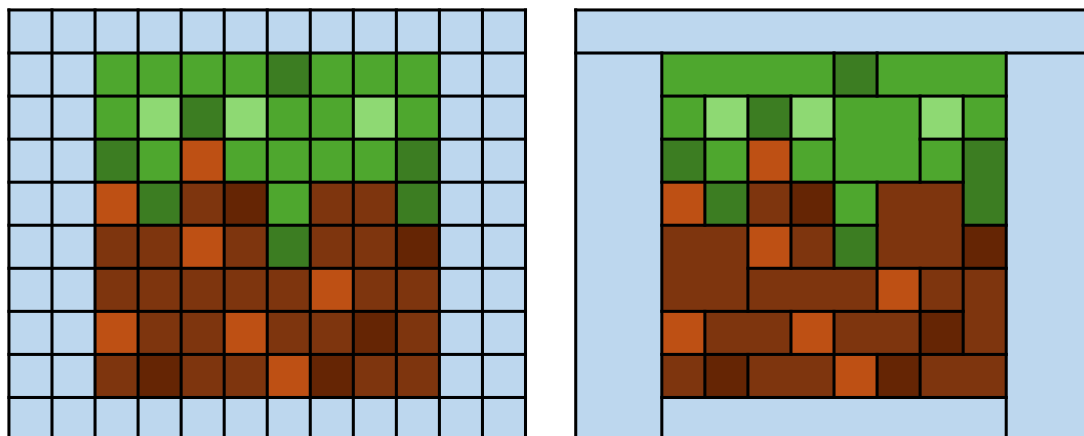
5.1.2 Zakreslování buněk

Množství výpočtů je pouze jedna ze dvou hlavních složek běhu programu. Další podstatnou částí je práce s grafickým rozhraním – s buňkami. Operace zakreslení jedné buňky (nebo obdélníku buněk) je časově náročnější operací než matematické operace. V současné verzi programu je možnost vypnout aktualizaci grafického rozhraní během zakreslování buněk a následně ji zapnout až po dokončení vykreslování. Avšak v Excelu trvá tento přechod déle s rostoucím počtem zakreslených operací, protože jednotlivé změny vyžadují aktualizaci grafického rozhraní a vykreslení nových buněk. To může vést ke zpoždění, zejména při práci s velkým množstvím dat nebo při složitých operacích s formátováním buněk. Toto chování může dokonce vést k pádu aplikace z důvodu přetečení čísel nebo přeplnění paměti s náhodným přístupem (*RAM*).

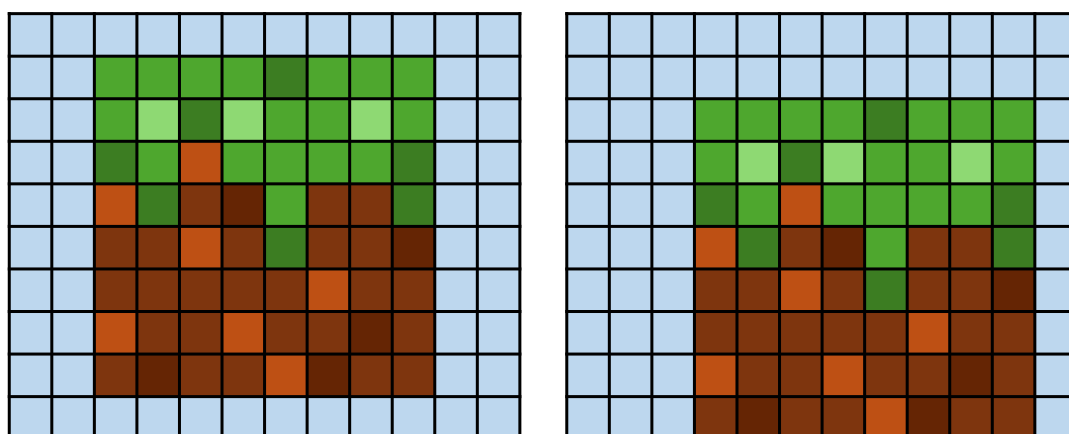
Program momentálně zakresluje i strany, které nejsou vidět, což vede k opakované změně barev pixelů během vykreslování jednoho snímku. Toto lze vyřešit například vytvořením 2D pole, do kterého by se tyto barvy zapisovaly. Poté by program mohl zakreslovat barvy z tohoto pole do buněk až po dokončení vykreslení všech barev. Takový přístup by snížil neefektivitu při vykreslování a minimalizoval opakovanou změnu barev pixelů během jednoho daného snímku.

Implementováním této funkce by ovšem znamenalo zhoršení rychlosti, kvůli nemožnosti vykreslovat buňky po obdélnících, namísto po jednotlivých buňkách. Proto je potřeba k tomu implementovat funkci, která by seskupila jednotlivé pixely do obdélníků (obr. 5.2).

Podobně jako u některých video formátů, které efektivně ukládají data tak, že uloží pouze první snímek a dále pouze změny pixelů od předchozích snímků (obr. 5.3), dochází k výrazné redukci velikosti souboru. Tímto procesem lze také urychlit vykreslování snímků, protože program nebude muset znova vykreslovat buňky.

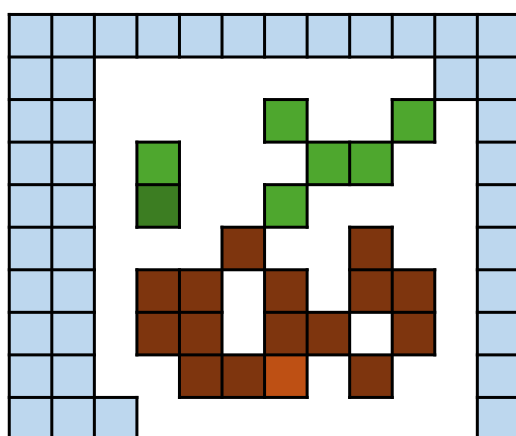


obr. 5.2: vybarvování buněk – vlevo po jednotlivých buněk, napravo po obdélnících

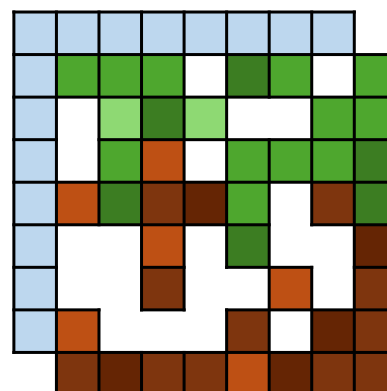


Předchozí snímek

Vykreslovaný snímek



Průnik stejných buněk obou snímků



Rozdíl buněk snímků
(to co bude zakresleno)

obr. 5.3: vybarvování pouze změněných buněk – vpravo dole

5.2 VYUŽITÍ

5.2.1 Edukační

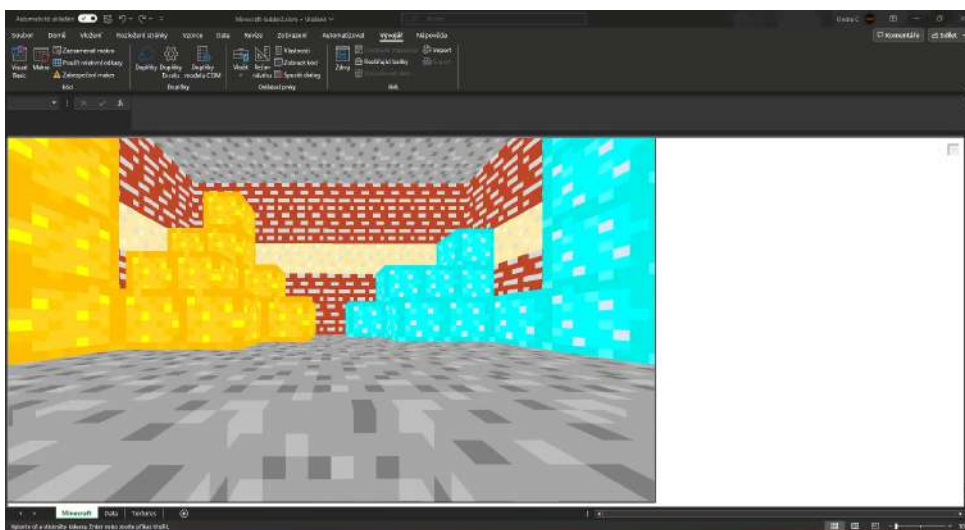
Jedna z nejpodstatnějších využití práce je k edukaci. Celá tato práce dokumentuje a vysvětluje principy pohybování a otáčení pomocí lineárního zobrazení v soustavě 3D, převádění souřadnic do 2D pomocí perspektivní projekce, vysvětluje nejpodstatnější části kódu a jejich funkce a optimalizaci programu. Práce obsahuje spoustu klíčových konceptů myšlenek i věcí, na které si je třeba dávat pozor při práci v 3D enginu.

5.2.2 3D engine

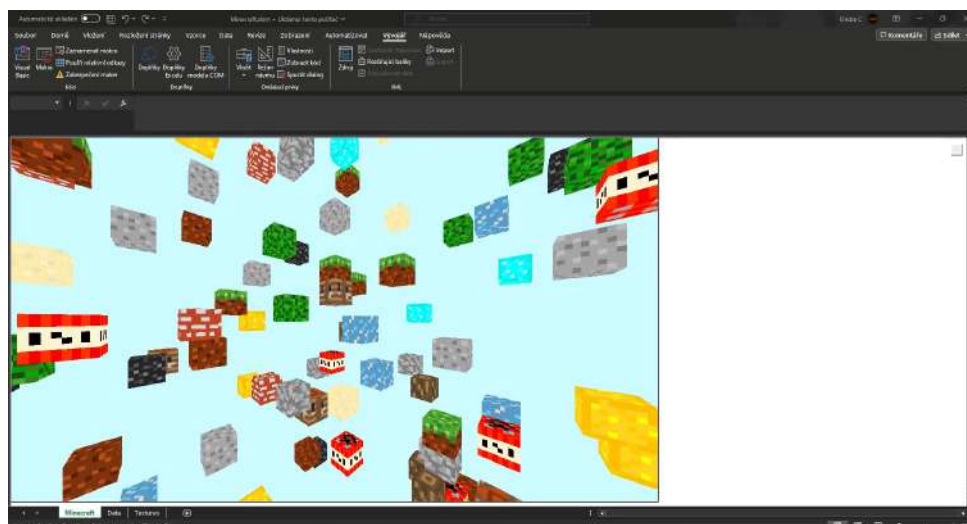
Další zásadní přínos práce je implementace 3D prostředí, které umí pracovat s otexturovanými čtverci, a je poměrně jednoduché přepsat program tak, aby byl schopný pracovat i s trojúhelníky, jak je běžné u většiny 3D enginů.

5.2.3 Animace

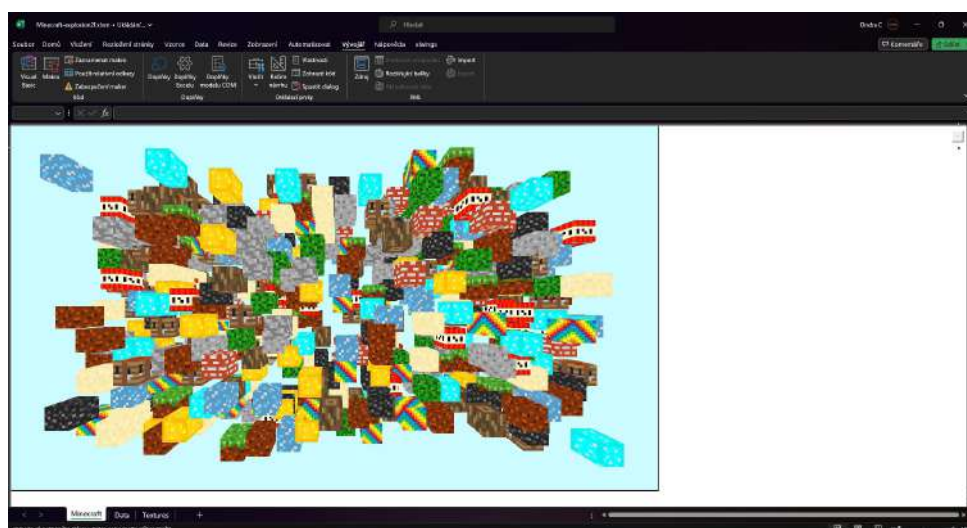
Třetí hlavní možností, jak využít práci, je k animaci. Součástí programu je balíček, který umožňuje automatizaci procesu pohybu a otáčení kolem objektu nebo modifikaci souřadnic krychlí. Výstupem jsou sady obrázků, které představují jednotlivé snímky animace a lze je poté zkombinovat pomocí externích programů do souborů .gif (k vytvoření samotné animace). Jednotlivé výstupy jsou v příloze spolu s nástroji a instrukcemi pro jejich používání. (příklady: obr. 5.4, obr. 5.5 a obr. 5.6)



obr. 5.4: příklad animace – 1x snímek z animace stoupání nahoru v budově



obr. 5.5: příklad animace – 1x snímek z animace padajícího hráče kolem krychlí



obr. 5.6: příklad animace – 1x snímek z animace roztržnění krychlí do prostoru a zpět

5.3 MOŽNOSTI POKRAČOVÁNÍ

Jelikož se jedná o kostru zobrazování krychlí ve 3D, nabízí se možnost pokračování vývojem na portu pro počítačovou hru Minecraft. Avšak před tím je zapotřebí vyřešit několik problémů.

K naprogramování základů této hry je nejprve třeba vytvořit několik herních funkcí. To zahrnuje simulaci jednoduché fyziky, jako je například detekce kolizí s bloky, implementace gravitace a simulace světla. Takový přístup by znamenal, že vykreslování snímků by již nebylo podmíněno pohybem hráče, ale snímky by se počítaly neustále. Proto je nutné primárně vyřešit problém stability Excelu. Další základní herní mechanikou je inventář a vyrábění předmětů, což by bylo možné realizovat pomocí tlačítek a dalších klávesových

zkratek. Důležitou součástí by také mělo být pokládání a těžení bloků. Program je již připraven na tuto implementaci, neboť ukládá vzdálenosti jednotlivých krychlí od hráče seřazené podle velikosti, a stačilo by pouze ověřit, zda se blok nachází uprostřed obrazovky.

V neposlední řadě by uživatelům zjednodušilo přidávání nových tvarů parametrizace bodů. V oddíle 3.6.3 bylo vysvětleno, jak přidávat nová tělesa, ale to vyžaduje změny a doplnění dalších možností do podmínek ve třech různých částech programu. Kvůli nemožnosti přehledného formátování kolekce (ve Visual Basic for Application) by bylo nutné využít externího souboru například ve formátu *JSON*, kde by byly uloženy veškeré parametry pro jednotlivé bloky. Vytvoření funkce, která by byla schopna načíst jednotlivé datové typy z *JSON* souboru, však přináší s sebou velké množství problémů vzhledem k povaze objektově orientovaného jazyka VBA.

6 ZÁVĚR

Tato ročníková práce měla za cíl implementovat program pro renderování 3D krychlí pomocí 2D zobrazení do buněk v prostředí Microsoft Excelu s využitím jazyka Visual Basic for Applications.

V první části se práce zabývala teorií počítání souřadnic na základě otáčení kamery a pohybu hráče. Další částí byla samotná implementace problému, kde jsem podrobně popsal každou funkci, která za programem stojí. Poslední část se zabývala problémy prostředí Excelu, který znemožnil dosažení uspokojivých výsledků ohledně rychlosti výpočtu snímků. Tento problém byl následně řešen pomocí implementace algoritmů, jež sloužily k optimalizaci programu a redukci nutných výpočtů.

Závěrem lze proto říct, že Excel není vhodné prostředí k implementaci jakéhokoliv programu fungujícím na bázi 3D prostředí. Celkově informace v této práci mohou posloužit pro uživatele jako náhled do základů fungování třidimenzionálních prostředí. Program má také prostory pro vylepšení, a to jak z hlediska optimalizací, tak i z hlediska uživatelského rozhraní.

LITERATURA

- [1] ROBOVÁ, J., a spol. (2010). *Soustava souřadnic v prostoru*. Katedra didaktiky matematiky, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze. [online]. Dostupné z: https://www.karlin.mff.cuni.cz/~portal/analyticka_geometrie/souradnice.php?kapitola=soustavaSouradnicP.
- [2] HORČÍK, R. (2009). *Perspektivní projekce*. Ústav informatiky Akademie věd ČR. [online]. Dostupné z: <https://uivty.cs.cas.cz/~horcik/Teaching/applications/node4.html>.
- [3] TICHOTA, M. (2009). *Perspektiva jako matematický model objektivu*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2. [online]. Dostupné z: <https://home.zcu.cz/~mikaMM/Galerie%20studentskych%20prac%20MM/2009/Tichota-Perspektiva%20v%20PG.pdf>.
- [4] IVANOV, O., et al. (2011). *Applications and Experiences of Quality Control*. InTech. Research about New Predictive-Maintenance Methodology using VBA for Marine Engineering Applications. José A. Orosa, Angel M. Costa a Rafael Santos, University of A Coruña, Spain, 399–410. [online]. Dostupné z: <https://cdn.intechopen.com/pdfs/14853.pdf>.
- [5] UNITY TECHNOLOGIES (2023). *Understanding the View Frustum*. Unity Documentation. [online]. Dostupné z: <https://docs.unity3d.com/Manual/UnderstandingFrustum.html>.
- [6] LIPA, M., & RAJMIC, P. (2019). *Rasterizace úsečky pomocí Bresenhamova algoritmu – applet*. Ústav telekomunikací, FEKT, VUT v Brně. [online]. Dostupné z: https://www.utko.fekt.vut.cz/~rajmic/applets/Bresenham_line/line.html.
- [7] JOHNSON, M. (2018). *9 quick tips to improve your VBA macro performance*. Microsoft Tech Community, část 6. [online]. Dostupné z: <https://techcommunity.microsoft.com/t5/excel/9-quick-tips-to-improve-your-vba-macro-performance/m-p/173687>.
- [8] HOARE, C. A. R. (1962). *Quicksort*. The Computer Journal, 11–12. [online]. Dostupné z: <https://academic.oup.com/comjnl/article-pdf/5/1/10/1111445/050010.pdf>.

- [9] KRÁL, M. (2012). *Excel VBA*. Computer Press, 11–16, 442, 478. [online]. Dostupné z: <https://books.google.cz/books?id=khu2DwAAQBAJ>.
- [10] DANZIGER, P. (2016). *Big O Notation*. 1–6. [online]. Dostupné z: <https://www.cs.ryerson.ca/~mth210/Handouts/PD/bigO.pdf>.
- [11] MUNAWAR, K. (2021). *Video Aspect Ratios: A Definitive Guide*. [online]. Dostupné z: <https://motioncue.com/video-aspect-ratios>.
- [12] HAAVERTZ, M. (2023). *Does CPU Affect FPS*. Kingston College London. [online]. Dostupné z: <https://kingstoncollege.org/does-cpu-affect-fps>.
- [13] Příspěvatelé Wikipedie (2021). *Lineární zobrazení [online]*. Wikipedie: Otevřená encyklopedie. [online]. [citováno 22. 03. 2024]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Line%C3%A1rn%C3%AD_zobrazen%C3%AD&oldid=20724346.
- [14] Příspěvatelé Wikipedie (2023). *Perspektivní promítání*. Wikipedie: Otevřená encyklopedie. [online]. [citováno 22. 03. 2024] Dostupné z: https://cs.wikipedia.org/w/index.php?title=Perspektivn%C3%AD_prom%C3%ADt%C3%A1n%C3%AD&oldid=23155156.

SEZNAM POJMŮ

- **Tabulkový procesor** – Aplikace pro manipulaci s tab. daty (Microsoft Excel).
- **3D** – Trojrozměrný prostor tvořený osami X, Y, Z.
- **2D** – Dvourozměrný prostor definovaný dvojicí os (například X, Y).
- **Hráč** – Pozice, z níž je pozorována zbytek scény.
- **Kamera** – Směr, ze kterého se hráč dívá.
- **VBA** – Visual Basic for Applications (programovací jazyk používaný v aplikacích Microsoft Office pro automatizaci úkolů, například v Excelu k interakci s buňkami).
- **Snímek** – Jedna statická instance scény tvořená vybarvenými buňkami.
- **Block** – Označení pro objekt (v základním složení krychle, která má střed a 6 stran).
- **Side** – Označení pro jednu stranu, která se skládá ze 4 vrcholů (v základním složení čtverec) a tvoří Block.
- **Pixel** – Označení pro jednotku textury s jednotnou barvou, tvořící jednotlivé strany (v základním složení má 1 Side 8x8 pixelů).
- **Buňka** – Jednotka v tabulkovém procesoru obsahující barvu a hodnotu, dohromady tvoří obrazovku.
- **Kolekce** – Skupina souvisejících objektů nebo prvků; v prostředí VBA označení pro list nebo slovník.
- **Lineární zobrazení (Lineární transformace)** – Zobrazení, které zachovává vektorové operace sčítání a násobení skalárem mezi vektorovými prostory X a Y [13].
- **Perspektivní projekce (Perspektivní promítání)** – Způsob promítání, kde promítací přímky vycházejí ze společného bodu (kamery), který nesmí ležet v průmětně, a není zachována rovnoběžnost paprsku vycházejících z kamery [14].
- **Landauova notace (Big O notation)** – Zápís k odhadu složitosti algoritmů.
- **Herní engine** – Softwarový systém pro vývoj a provoz videoher.
- **GPU** – Grafický procesor, specializovaný hardware pro zpracování grafiky a urychlení výpočtů.

SEZNAM OBRÁZKŮ

2.1	pravotočivé osy X, Y, Z	5
2.2	otáčení bodu A o úhel θ v kartézské soustavě souřadnic roviny XY	6
2.3	pravoúhlý trojúhelník s úhlem β u vrcholu B	8
2.4	otáčení bodu A o úhel θ (<i>yaw</i>) v případě kamery otáčení kolem osy Y . . .	10
2.5	otáčení bodu A o úhel θ (<i>pitch</i>) v případě kamery otáčení kolem osy X . .	11
2.6	perspektivní zobrazení čtverců na obrazovku, pohled shora – převedeno do 2D, pro lepší orientaci	13
3.1	uživatelské rozhraní programu v prostředí Excelu	16
3.2	seznam textur krychlí v 3. sešitu v Excelu	18
3.3	vizualizace aplikace textur – vlevo 1x Side, napravo 64x Pixel	22
3.4	zorné pole hráče, pohled shora – převedeno do 2D, pro lepší orientaci . . .	24
3.5	Bresenhamův algoritmus pro získávání buněk na základě přímky	25
4.1	závislost doby výpočtu jednoho snímku na celkovém počtu vykreslovaných krychlí (optimalizovaný kód)	34
4.2	závislost doby výpočtu jednoho snímku na počtu vykreslovaných buněk (optimalizovaný kód)	35
4.3	graf základní struktury celého programu	40
4.4	příklad vykreslení snímku pomocí programu	40
5.1	graf základní struktury programu s výpočty v <i>OpenCL</i>	42
5.2	vybarvování buněk – vlevo po jednotlivých buněk, napravo po obdélnících	43

5.3	vybarvování pouze změněných buněk – vpravo dole	43
5.4	příklad animace – 1x snímek z animace stoupání nahoru v budově	44
5.5	příklad animace – 1x snímek z animace padajícího hráče kolem krychlí . . .	45
5.6	příklad animace – 1x snímek z animace roztržštění krychlí do prostoru a zpět	45

SEZNAM TABULEK

4.1	doba výpočtu jednoho snímku v závislosti na celkovém počtu krychlí (optimalizovaný kód)	35
4.2	doba výpočtu jednoho snímku v závislosti na celkovém počtu vykreslovaných buněk (optimalizovaný kód)	36
4.3	doba trvání průměrného vykreslení obrázku v sekundách	39
4.4	poměr optimalizací vůči programu bez žádných optimalizací vztaženo k bez optimalizací v procentech	39
4.5	poměr celkové optimalizace vůči konkrétní optimalizaci vztaženo k celkové optimalizaci v procentech	39
6.1	bez optimalizace (oddíl 4.3.3)	56
6.2	zorné pole hráče (oddíl 4.3.4.1)	56
6.3	počet stran krychle (oddíl 4.3.4.2)	56
6.4	duplicitní strany (oddíl 4.3.4.3)	57
6.5	vykreslování po řádcích (oddíl 4.3.4.4)	57
6.6	veškeré optimalizace (oddíl 4.3.5)	57

SEZNAM UKÁZEK KÓDŮ

2.1	perspektivní zobrazení bodů; názvy proměnných upraveny	12
2.2	perspektivní zobrazení bodů; upraveno	14
3.1	aplikace textury každé ze stran; upraveno	22
3.2	převod souřadnic ze 3D do 2D; upraveno	23
3.3	kontrola, jestli se nachází bod uvnitř zorného pole; upraveno	25
3.4	pohybování hráče nahoru, dolů; upraveno	28
3.5	pohybování hráče doleva, doprava; upraveno	28
3.6	pohybování hráče dopředu, dozadu; upraveno	29
3.7	otáčení kamery nahoru, dolů; upraveno	29
3.8	otáčení kamery doleva, doprava; upraveno	30
3.9	aplikace průhledné buňky; upraveno	31

PŘÍLOHY

1. Zdrojový Kód – Složka, která obsahuje čistě zdrojový kód.

- Třídy:

1. **Block.cls**
2. **Calculations.cls**
3. **Game**
4. **Pixel.cls**
5. **Player.cls**
6. **Side.cls**
7. **Stats.cls**
8. **Textures.cls**

- Moduly:

1. **Functions.bas**
2. **Geometry.bas**
3. **Keys.bas**
4. **Main.bas**
5. **Visual.bas**

2. Animace – Složka, obsahující 5 výsledných video animací a modul funkcí:

1. **cubeRotation.mp4** – Otáčení jedné krychle podle její osy Y .
2. **explosion.mp4** – 300 krychlí vycházející z jednoho bodu po osách X , Y
3. **fallingPitAround.mp4** – Pohybování hráče po ose Y dolů kolem krychlí.
4. **islandRotation.mp4** – Otáčení modelu ostrova podle jeho osy Y .
5. **ladder.mp4** – Pohybování hráče po ose Y nahoru, pře ním model budovy.
6. **_addToMain.bas** – Soubor obsahující instrukce a funkce sloužící k animaci.

3. zdrojovýKód.pdf – celý zdrojový kód v jednom .pdf souboru.

4. CubesRendering.xlsm – Výsledný soubor práce v Microsoft Excelu schopný vykreslovat snímky. Zdrojový kód je součástí souboru.

Zdrojový kód je k dispozici také online na: [<https://github.com/ProfiPoint/minecraft-excel>]

TABULKY

6 tabulek naměřených dob vykreslování. 10 měření po 5 sadách pro každou optimalizaci. Měření vždy probíhalo s využitím všech optimalizací (oddíl 4.3.4) s vynecháním dané optimalizace, **v sekundách [s]**; červeně chybné, nebo záporné).

	Uvnitř jedné krychle	27 sousedících krychlí	50 náhodných krychlí	50 krych. bez zakreslní	50 krych. pouze zakres.
1	83,421	462,797	3193,812	2512,327	681,485
2	85,039	461,758	945,505	721,516	223,989
3	82,496	461,221	1141,775	915,820	225,955
4	83,691	464,545	912,871	718,750	194,121
5	83,843	462,612	-	-	-
6	83,211	461,657	1274,027	987,548	286,479
7	82,793	461,680	955,753	751,949	203,804
8	83,687	462,976	2154,002	1885,363	268,639
9	82,981	462,675	1362,336	1152,305	210,031
10	84,652	461,990	845,172	659,086	186,086

tab. 6.1: bez optimalizace (oddíl 4.3.3)

	Uvnitř jedné krychle	27 sousedících krychlí	50 náhodných krychlí	50 krych. bez zakreslní	50 krych. pouze zakres.
1	56,656	22,5	1410,147	1397,542	12,605
2	57,226	22,54	652,652	646,417	6,235
3	56,735	22,359	761,587	759,532	2,055
4	56,367	22,875	-	-	-
5	57,039	22,305	487,726	491,552	-3,826
6	56,867	22,258	568,121	554,235	13,886
7	56,672	22,367	455,133	462,042	-6,909
8	57,133	22,399	1163,438	1152,523	10,915
9	57,219	22,304	647,773	654,61	-6,837
10	56,805	22,39	328,485	322,004	6,481

tab. 6.2: zorné pole hráče (oddíl 4.3.4.1)

	Uvnitř jedné krychle	27 sousedících krychlí	50 náhodných krychlí		50 krych. pouze zakres.
1	5,297	39,547	79,816	65,411	14,405
2	5,230	39,430	47,521	35,606	11,915
3	5,043	39,352	45,598	40,851	4,747
4	5,363	38,945	39,344	35,164	4,180
5	5,293	38,977	51,234	45,817	5,417
6	5,160	40,078	39,282	35,418	3,864
7	5,449	40,797	59,891	57,078	2,813
8	5,330	38,890	43,863	41,153	2,710
9	5,105	39,500	55,637	55,543	0,094
10	5,179	39,876	48,844	42,086	6,758

tab. 6.3: počet stran krychle (oddíl 4.3.4.2)

	Uvnitř jedné krychle	27 sousedících krychlí	50 náhodných krychlí	50 krych. pouze zakres.	
1	5,375	57,617	53,391	45,273	8,118
2	6,211	57,586	28,875	26,375	2,500
3	5,289	58,000	34,446	34,274	0,172
4	5,856	57,938	25,281	21,899	3,382
5	5,486	57,922	41,750	35,453	6,297
6	6,180	58,391	27,758	23,969	3,789
7	5,185	57,625	46,461	38,648	7,813
8	5,732	57,687	29,711	26,484	3,227
9	5,168	57,984	39,250	29,062	10,188
10	5,207	57,843	33,234	24,086	9,148

tab. 6.4: duplicitní strany (oddíl 4.3.4.3)

	Uvnitř jedné krychle	27 sousedících krychlí	50 náhodných krychlí	50 krych. pouze zakres.	
1	33,648	35,113	153,578	40,625	112,953
2	33,402	35,082	75,414	22,899	52,515
3	34,453	35,035	92,812	27,789	65,023
4	33,930	35,118	59,055	19,328	39,727
5	34,082	35,082	125,023	31,812	93,211
6	33,664	35,097	75,523	21,109	54,414
7	33,778	35,180	140,414	34,368	106,046
8	33,516	35,035	85,375	22,757	62,618
9	34,025	35,152	106,274	29,039	77,235
10	33,655	35,265	68,672	24,211	44,461

tab. 6.5: vykreslování po řádcích (oddíl 4.3.4.4)

	Uvnitř jedné krychle	27 sousedících krychlí	50 náhodných krychlí	50 krych. pouze zakres.	
1	5,251	22,769	46,754	42,180	4,574
2	5,494	22,945	25,953	23,922	2,031
3	5,884	22,719	24,723	22,986	1,737
4	5,484	22,889	22,539	20,321	2,218
5	5,748	21,707	40,305	34,313	5,992
6	5,052	21,726	24,372	22,242	2,130
7	5,419	21,769	39,367	35,804	3,563
8	5,204	21,773	25,782	24,250	1,532
9	5,429	21,780	33,250	30,593	2,657
10	5,405	21,772	28,207	25,375	2,832

tab. 6.6: veškeré optimalizace (oddíl 4.3.5)