Středoškolská technika 2025

Setkání a prezentace prací středoškolských studentů na ČVUT

Jednoduchý optický skener

Lukáš Kotulán

SPŠ a VOŠ Brno, Sokolská,

příspěvková organizace

Sokolská 1, 602 00 Brno

Poděkování

Chtěl bych tímto poděkovat svému vedoucímu práce, panu Ing. Jaroslavu Nesvadbovi, a také panu Horňákovi za cenné rady a poskytnuté informace. Také mým přátelům za nedocenitelnou pomoc a zpětnou vazbu.

Anotace

Ve své práci jsem realizoval jednoduchý optický 3D skener na základě odrazu světla laseru do optického sensoru. Tento skener dokáže digitalizovat jednoduchý předmět do formy bodů ve 3D prostoru. Finálním výsledkem projektu je potom zobrazení těchto bodů.

Využil jsem mikropočítač Arduino programovaný v příslušném jazyce. Snímky jsou zpracovány programem v jazyce Python v prostředí Visual Studio Code. Získané body jsou zobrazeny v aplikaci, kterou jsem vytvořil v prostředí Unreal Engine 5 pomocí kombinace C++ a vizuálního skriptování.

Klíčová slova

laser, skener, Arduino, 3D tisk, Unreal Engine

Annotation

In my project I developed a simple optical 3D scanner based on laser line reflection. This scanner is able to digitize a simple object into a series of points in 3D space. These points, once visualized, form the final result of the project.

I have used the Arduino microcontroller programmed in its own proprietary language. The images are processed in a Python program created in the Visual Studio Code environment. The acquired points are displayed in an application I have made in Unreal Engine 5 using a combination of C++ and visual scripting.

Keywords

laser, scanner, Arduino, 3D printing, Unreal Engine

Obsah

1. Zadání

Cílem práce je vytvořit jednoduchý optický 3D skener na základě odrazu světla laseru do optického sensoru. Tento skener dokáže digitalizovat jednoduchý předmět do formy bodů ve 3D prostoru. Finálním výsledkem projektu je potom zobrazení těchto bodů.

2. Úvod

Nápad na tento projekt vznikl během mé školní praxe ve firmě ABB. Tam jsem během dvou týdnu pomáhal sestavováním relátek, ovládacích panelů, pojistek a podobně. Jednoho dne zaměstnanec sedící naproti mně měl problém, že se mu ve vrtačce zlomil kus plastu, který sloužil ke spínání nástroje. Nadřízený navrhl, že na firemní 3D tiskárně vytisknou nový a povolali právě mě, abych ten kus plastu změřil a vymodeloval v CAD programu SolidWorks. Vše proběhlo úspěšně a po celé věci vznikla na pracovišti diskuse proč už neexistuje nějaký jednoduchý, přesný, a hlavně levný 3D skener pro podobné použití ve strojírenství. Ten by v této situaci posloužil perfektně. Jelikož praxe byla velice jednoduchá a vesměs monotónní, následující dny jsem přemýšlel nad tím, jak jednoduše naskenovat nějaký objekt.

Hlavní koncept tohoto projektu má velmi podobnou podobu toho, co jsem tehdy na jaře na praxích vymyslel, i když později během realizace jsem musel čelit mnoha neočekávaným výzvám, jak už to u takových projektů bývá.

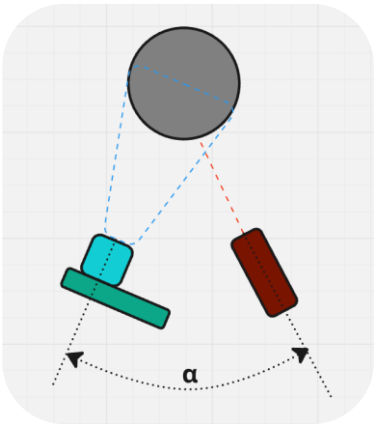
Co je na tom celém ovšem nejzajímavější je skutečnost, že některé koncepty a řešení problémů v tomto projektu jsem nejprve samostatně vymyslel, a až později jsem zjistil že tato řešení už jsou známá a používaná. Například odraz světla při skenování lesklých objektů může negativně ovlivnit výsledky. To lze vyřešit práškovým sprejem, který pokryje předměty tenkou matnou vrstvou, aby měly perfektní vlastnosti pro skenování. Nebo určitá technologie pro vykreslování velkého množství bodů v prostoru. Tyto problémy jsem nejdříve vyřešil sám a až potom jsem zjistil že přesně tato řešení už existují a jsou veřejně adoptována. Tato skutečnost mi dost zvýšila sebevědomí, že dokážu vhodně a realisticky řešit problémy tak, aby šly aplikovat v praxi.

3. Hardware

Původně jsem se snažil navrhnout mechanismus, pomocí kterého by se hlavní části do sebe „zaklapnuly“. Ovšem tento způsob se ukázal nespolehlivým, protože čím větší součásti tisknete, tím stoupá jejich nepřesnost. Nakonec jsem se rozhodl, že do třech vnějších částí během tisku vložím magnety. Toto řešení eliminovalo skoro jakoukoliv potřebu pro přesnost a velice zjednodušilo montáž a demontáž skeneru.

3.1. Optika

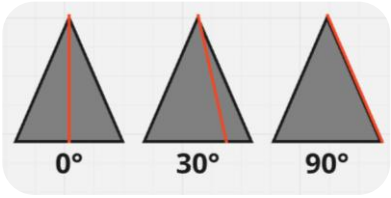
Na přiloženém obrázku je šedou barvou znázorněn skenovaný objekt, červenou laserová dioda a modrou kamera.



Obrázek 3: Zjednodušený pohled ze shora

Obrázek 1: Pohled z kamery: Tvar laseru se mění podle skenovaného objektu

Celý proces skenování závisí na laserovém obrysu, který se odrazí od objektu do kamery. Při konstrukci je důležité zvolit správný úhel mezi kamerou a laserem **α**. Pokud je úhel **0°**, laserový obrys je přímo rovnoběžný s kamerou a z pohledu kamery bude vždy rovný. Proto nelze nijak rozpoznat tvar objektu. Pokud je úhel **90°** a větší, tak je laserový obrys kolmý na kameru a není vidět. Ovšem čím větší úhel, tím výraznější je změna tvaru obrysu. Nakonec jsem s ohledem na další konstrukční důvody zvolil **30°**.



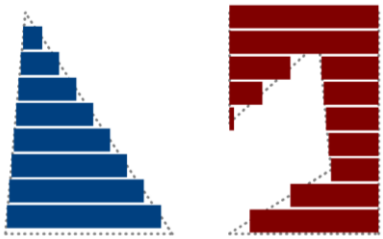
Obrázek 2: Pohled z kamery na obrys při různých úhlech polohy laseru

Aby kamera mohla zachytit obrys objektu po celé výšce, je potřeba aby laserová dioda promítala svislou čáru. Původně jsem měl v plánu zakoupit tradiční laserové ukazovátko, které vytváří laserovou tečku a poté vyrobit nebo koupit čočku, která laserovou projekci protáhne do čáry. Na internetu jsem zjistil, že tato čočka je velice jednoduchá na získání, jedná se totiž o normální válec. Laserová čára pak vznikne rovnoběžně s čelem válce. Na soustruhu jsem z čistého plastu vyrobil přesně takovou čočku, ale při vybírání laseru na internetu jsem zjistil, že se běžně prodávají laserové diody už s čočkou, která byla navíc kvalitnější a lépe rozprostřela světlo rovnoměrně po celé čáře. Navíc toto zařízení stalo 60 korun a jeho světelný výkon naprosto stačí pro moje účely.

Rozhodl jsem se, že budu pracovat pouze se skenováním černých matných objektů, protože tím eliminuji co nejvíce odrazu světla do všech okolních směrů a mezi osvětleným obrysem a neosvětleným zbytkem objektu bude co největší kontrast.

3.2. 3D tisk

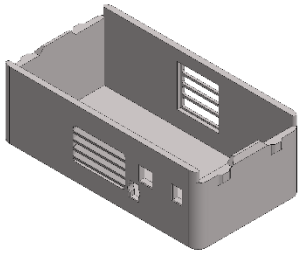
Většinu celého zařízení tvoří 3D tisknuté díly. Tisknul jsem na vlastní tiskárně a musím uznat, že to byla jedna z nejlepších koupí, co jsem mohl udělat. Díky tiskárně můžu přivést do reality skoro jakýkoliv tvar a otevírá to obrovské možnosti pro osobní projekty. Samozřejmě, že tiskárny mají svůj limit. Největším omezením je složitost tvarů v ose Z (nahoru a dolů), protože tiskárna vytváří objekty tak, že klade jednotlivé vrstvy plastu na sebe. Proto nesmí vzniknout vrstvy, které pod sebou nic nemají, protože tekutý plast by jednoduše stekl dolů a náš tisknutý tvar by byl deformovaný.



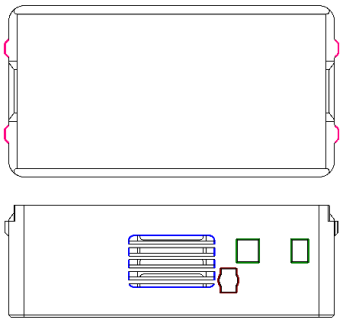
Obrázek 4: Pohled tisknutých vrstev z boku: Vhodný (nalevo) a nevhodný (napravo) tvar pro 3D tisk

3.2.1. Krunýř

Největší tisknutý díl je takzvaný „**krunýř**“. Tvarem se podobá jednoduché krabici a slouží k ochraně a podpoře všech komponentů. Dále se na ní nacházejí otvory na přívod a odvod vzduchu, pro spínač a pro konektory k Arduinu. Na horním okraji jsem vymodeloval drážky, do kterých se zaklapne víko. Na přiloženém nákrese můžete vidět skryté **magnety** – fialová, **otvor pro ventilaci** – modrá, **otvor pro vypínač** – červená, **otvor pro konektory Arduina** – zelená



Obrázek 5: Model krunýře

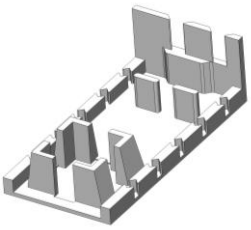


Obrázek 6: Nákres krunýře

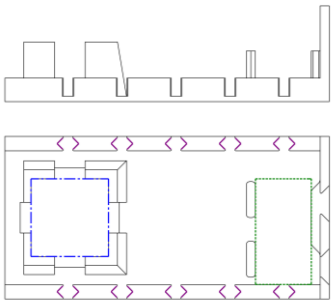
Kvůli své velikosti byl tento díl nejtěžší na výrobu. Aby řádně chránil celé zařízení, zprvu jsem se snažil díl vytisknout z plastu **ABS**. Tento plast je velmi pevný, tvrdý a odolný vůči UV záření a vnějším vlivům. Bohužel je ale také extrémně náchylný na změnu teploty. Kvůli tomu se neustále během tisku deformoval a po dokončení tisku se objevily praskliny mezi jednotlivými vrstvami. Materiál **PLA** byla další možnost. Tento plast není tak odolný, zato má perfektní vlastnosti k tisku. Kdybych měl tento filament v červené barvě, bez váhání bych ho zvolil. Ale jelikož jsem chtěl udržet určitou úroveň estetiky, zvolil jsem jiný červený plast, který jsem měl k dispozici. Tím mohl být buď **ABS** nebo **PETG**. Z **ABS** se mi tisk nedařil, a tak jsem zvolil **PETG**. Tento filament jsem měl stejně jako **ABS** v červené barvě. **PETG** materiál není tak pevný, ale je pevnější než **PLA** a je pružný, díky čemuž velmi dobře absorbuje energii vnějšího působení. Bohužel neustálé technické problémy s tiskárnou mě donutily opustit estetiku a krunýř jsem vytiskl z bílého **PLA** filamentu, jelikož v tomto momentě bylo prioritou projekt dokončit.

3.2.2. Kostra

Motor, Arduino, baterie, pojistka a další části leží v takzvané „kostře“. Jedná se o druhý největší tisknutý model uvnitř krunýře, který slouží jako opěra, na které drží většina komponentů na pevně svém místě. Na nákrese vidíme místo pro **motor** – modrá, **místo pro baterii** – zelená, **univerzální otvory** – fialová.



Obrázek 7: Model kostry



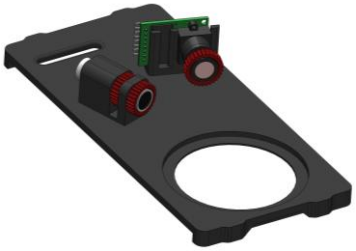
Obrázek 8: Nákres kostry

Počítal jsem s tím, že pokud bych všechna uchycení zakomponoval do jednoho modelu, tento díl by musel podstoupit velkým množstvím změn, protože ideální způsob přichycení, tolerance, velikosti a podobně se velice špatně odhadují. Proto jsem konstrukci zjednodušil pouze na to nejdůležitější: přihrádka pro baterii, přihrádka pro motor. K uchycení všeho ostatního jsem po stranách navrhl **prismové otvory**, do kterých budu moct přichytit cokoliv co bude potřeba.

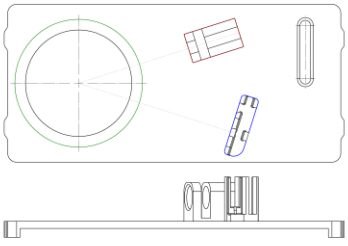
V aktuální verzi skeneru v těchto otvorech drží kryt na pojistku, držák větráku a háčky pro organizaci vodičů.

3.2.3. Víko

Víko skeneru je třetí největší díl, na kterém se nachází držák pro kameru a laser a taktéž otvor pro vodiče a otvor, do kterého patří otočná plocha pro skenovaný objekt. Na kratších stranách jsou také celkově čtyři magnety. Pomocí těchto magnetů víko drží na krunýři a zároveň na něm drží obou kryt. Na nákrese je zvýrazněn držák pro **laser** – červená, **držák kameru** – modrá, **otočná plocha** – zelená.



Obrázek 9: Model víka i s laserem a kamerou



Obrázek 10: Nákres víka

Při navrhování tohoto dílu jsem měl největší problém s navržením samotných držáků na laser a kameru. Ty totiž musí být navrhnuty tak, aby pevně a přesně držely elektroniku, ale zároveň nesmí nijak tlačit, nebo jinak na ně vyvíjet vnější sílu. Kromě toho také musí být jednoduché v ose Z (nahoru a dolů), protože celé víko i s držáky je tisknuté jako jeden kus. Samozřejmě, že by šlo tyto části tisknout samostatně, což by umožňovalo výměnu držáků a tím pádem rychlejší iteraci různých návrhů. To by ovšem vyžadovalo navrhnout nějaký mechanismus pro pevné a spolehlivé uchycení, který opět, musí být jednoduché v ose Z.

3.3. Kamera

Jako kameru jsem zvolil **Arducam OV5642**. Jedná se o integrovaný kamerový modul, který obsahuje samotný snímač, objektiv, paměť, rozhraní apod. Poskytuje snímání obrazu s nastavitelným rozlišením až **5** megapixelů (tedy rozlišení **2592 x 1944** pixelů). Díky kompaktním rozměrům a podpoře SPI komunikace je ideální volbou pro projekty s mikropočítači, například Arduino, ESP32, STM32 nebo Raspberry Pi.

Obrázek 11: Kamera Arducam OV5642 [1]

Jednou z klíčových vlastností tohoto modulu je podpora výstupních formátů obrazu včetně JPEG, BMP a RAW, což poskytuje flexibilitu při zpracování a ukládání obrazu. Komprese JPEG umožňuje efektivnější ukládání obrázků s menšími požadavky na paměť.

Kamera komunikuje s mikrokontrolery prostřednictvím dvou rozhraní: **SPI** (Serial Peripheral Interface) – používá se pro přenos obrazu a umožňuje kompatibilitu s širokou škálou mikrokontrolerů.
I2C (Inter-Integrated Circuit) – slouží k nastavování parametrů kamery, jako je rozlišení, jas, kontrast nebo režimy expozice.

Tento modul obsahuje interní 8MB FIFO buffer, který umožňuje dočasné ukládání snímků, aniž by zatěžoval hlavní procesor mikrokontroleru. FIFO je zkratkou pro „First In, First Out“. To znamená, že data, která jsou zapsána jako první budou přečtena jako první. To je zásadní vlastnost, která odlišuje tento kamerový modul od jednodušších modelů bez bufferu, jako jsou běžné VGA kamery. FIFO paměť umožňuje mikrokontroleru postupně číst obrazová data, čímž se minimalizují nároky na RAM paměť.

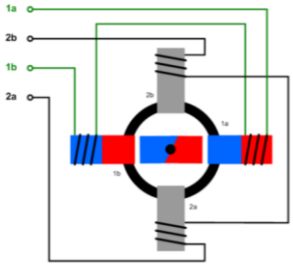
Obrázek 12: Snímek pořízený kamerou

Jedním z hlavních omezení tohoto modulu je relativně **pomalý** přenos dat přes SPI, což znamená, že není vhodný pro streamování videa v reálném čase. SPI rozhraní také není naprosto spolehlivé a může se stát, že se fotografie po cestě poškodí.

Další omezení vychází z pevných optických vlastností – kamera má fixní zaostření, takže nelze snadno měnit hloubku ostrosti. To znamená, že pro aplikace vyžadující proměnlivé zaostření (například sledování objektů na různých vzdálenostech) by bylo nutné přidat mechanický zaostřovací mechanismus nebo použít jiný kamerový modul s autofokusem. [1]

3.4. Motor

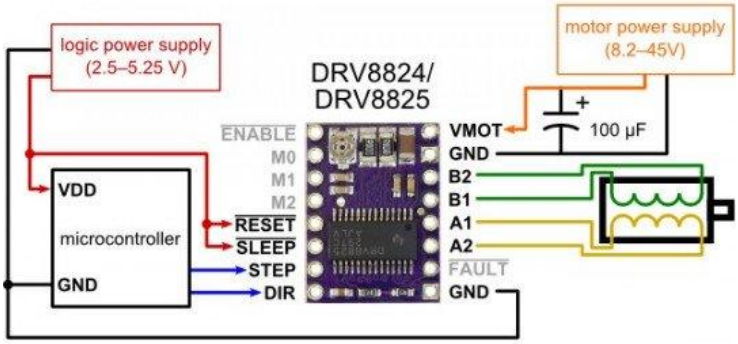
K otáčení objektu jsem zvolil jsem krokový motor NEMA 17, protože se jedná o ideální kompromis mezi cenou, velikostí a silou. Tento motor je bipolární. Je to speciální typ elektromotoru, který se používá pro velmi přesné řízení pohybu v aplikacích, jako jsou CNC stroje, 3D tiskárny a robotika. Bipolární krokový motor má dvě hlavní části: **stator**: obsahuje dvě cívky (vinutí), které jsou uspořádány tak, aby se střídaly. **Rotor**: obsahuje permanentní magnety nebo magnetizovaný materiál a otáčí se na základě elektromagnetických sil generovaných statorem.



Obrázek 13: Jednoduché schéma krokového motoru

Motor se pohybuje tak, že přepíná polaritu napájení cívek a tím nutí rotor, aby se pokaždé změně pootočil. Tomu se říká krok. Mnou zvolený motor má 200 kroků. Za napájení cívek je zodpovědný takzvaný driver. Jedná se o elektronický obvod, který na základě impulzů mikropočítače ovládá a střídá směr napájení do jednotlivých cívek.

Mnou použitý driver bere tři piny jako vstup. První pin – „**enable**“ určuje, zda je driver vůbec v provozu. Pokud do tohoto pinu nepouštím proud, pak driver neposílá žádné napětí do motoru a lze s ním volně hýbat. Pokud ale zapneme enable na logickou 1, pak se motor „zamkne“ v aktuální pozici a manuálně už s ním nemůžeme pohybovat. V tomto stavu ale motor spotřebovává energii a driver se zbytečně zahřívá, takže v mém projektu vždy zapnu enable, když chci s motorem pohnout a pak ho zase vypnu.



Obrázek 14: Zapojení driveru [4]

Další pin je takzvaný **dir**, zkratka pro direction – směr. Tento pin určuje směr, jakým se bude motor pohybovat. Pokud je v pinu logická nula, bude se pohybovat po směru hodinových ručiček, pokud bude v pinu logická jednička, pak se bude pohybovat v protisměru.

Nakonec ten nejdůležitější pin, **step – krok**, který je zodpovědný za samotný pohyb a jeho rychlost. Pin step přijímá impulzy, každý impulz znamená, že driver provede jeden krok. Takže čím rychleji posíláme impulzy do pinu step, tím rychleji se motor bude pohybovat. Toto má obrovskou výhodu, co se týče přesnosti, protože můžeme motor otočit o zvolený počet kroků a tím dosáhneme přesně takového úhlu, jaký potřebujeme.

Existuje také režim mikrokroků. Jedná se o to, že například místo celého kroku se napětí v cívkách změní jen o polovinu, nebo čtvrtinu, a tak se motor pohne třeba jen o čtvrtinu kroku. Toto umožňuje ještě větší přesnost, a hlavně plynulost pohybu. [2] [3] [4]

4. Software

Softwarová část projektu se skládá ze 3 samostatných programů, které jsem musel navrhnout tak, aby vzájemně bezchybně spolupracovaly.

4.1. Arduino

První fází je Arduino (více v následující podkapitole), které je uloženo v samotném 3D těle skeneru. Tento mikropočítač přijímá instrukce z, a posílá data do připojeného počítače. Podle přijatých instrukcí ovládá všechny prvky skeneru: laser, kameru a krokový motor. Tato první fáze slouží jako most mezi fyzickým světem a tím virtuálním, od tohoto bodu dál už se všechno odehrává v počítači.

4.1.1. Mikropočítač Arduino

Obrázek 15: Deska Arduino Uno [5]

Desky Arduino jsou mikropočítače, které umožňují snadnou interakci s různými elektronickými komponenty, jako jsou senzory, motory, LED diody nebo displeje. Desky mají několik různých typů. Např. Arduino Uno: nejpopulárnější model, ideální pro začátečníky. Arduino Mega: větší model, má více vstupů a výstupů pro složitější projekty. Nebo Arduino Nano: nejmenší model, pro malé a kompaktní obvody. Arduino je takzvaně open-source, což znamená že kód a schéma desky jsou veřejně dostupné, a tak si uživatelé můžou desky upravovat podle vlastních potřeb, nebo lze koupit neoficiální desky vyrobené za levnější cenu.

4.1.2. Napájení

Kromě otvoru na USB-B port je v krunýři také otvor na napájecí konektor Arduina, Ten ale není potřeba k běžné funkci skeneru. Arduino je totiž napájené z portu USB-B, kterým zároveň komunikuje s počítačem. Arduino pak napájí veškerou logiku. Například v driveru, samostatný elektronický obvod, který ovládá proud do krokového motoru, má dva zdroje elektrické energie. Jeden je hlavní, který putuje do motoru a druhý je logika, která ovládá ten první zdroj. Tato logika je napájená přímo z Arduina. Tato deska také napájí kameru a laser. Zdroj většího napětí, v našem případě 14 voltů, přináší akumulátor uložený pod Arduinem. Díky tomuto akumulátoru je skener závislý pouze na napájení logiky, a to je zajištěno vždy, když skener ovládáme, protože k němu stejně potřebujeme být připojení.

K účelům prezentace jsem vytvořil režim „standby“. V tomto režimu skener nepřijímá žádné instrukce a pouze pomalu bliká s laserem. V tomto případě není potřeba posílat žádná data do počítače, nebo s ním jakkoliv komunikovat, takže stačí skener připojit do zásuvky skrz dříve zmíněný napájecí port vedle USB-B.

4.1.3. Teorie programování Arduina

Arduino IDE (Integrated Development Environment) je programovací prostředí, ve kterém vývojáři vyvíjet kód do Arduino desky. Programování probíhá v jazyce, který je podobný jazyku C/C++, ale je upravený tak, aby byl přístupný i začátečníkům.

Programy pro Arduino se skládají ze dvou hlavních funkcí:
1. „**setup()**“: Tato funkce se spustí vždy jako první na začátku programu, zde probíhá nastavování proměnných a podobné přípravy.

2. „**loop()“**: Hlavní cyklus, který se neustále opakuje a obsahuje většinu programu např. interakci s hardwarem nebo komunikace s ostatními zařízeními.

[5]

4.1.4. Program pro skener

Setup():

V této části jsem nastavil veškerý hardware, který bude deska používat. Motor, na jakých pinech se ovládá, laser a na jakých pinech se nachází napájení, počáteční nastavení komunikace s počítačem skrz sériový port. Arduino má vestavěnou podporu sériové komunikace pomocí knihovny Serial. Programátor může inicializovat přenos pomocí:

```
void setup() {
  Serial.begin(9600);
}
```

Nejsložitější částí je nastavení kamery, jelikož má asi 8 pinů a její dokumentace je velice nedostačující, takže většina kódu pro kameru vznikla metodou pokus a omyl.

Loop():

V hlavní smyčce programu neustále běží komunikace s počítačem. Pokud přijmeme zprávu z počítače, porovnáme ji s příkazy v programu a pokud nějakou instrukci poznáme, provedeme ji. V kódu se vždy používají // pro poznámky.

```
String input = Serial.readString();     //Uložit přijaté instrukce
String instr = input.substring(0, 3);   //Uložit samostatně první 3 znaky


if (instr == "lon"){                    //Pokud instrukce je „lon“, zapnout laser
        digitalWrite(laserPin, HIGH);
}else if (instr == "lof"){
        digitalWrite(laserPin, LOW);
```

```
}else if (instr == "cpt"){
    captureImg();                   //Pořídit snímek a odeslat ho
    myCAM.CS_HIGH();
}else if (instr == "stn"){
    standby();                      //Zapnout „standby“ režim
}else if (instr == "mtr"){
    pos = input.substring(3, 7).toInt();    //Uložit další znaky z instrukce jako pozici
    moveToPos(pos);                         //Posunout se na pozici
}else if (instr == "tst"){
    Serial.println("This Is A Response");   //Testovací odpověď
}else if (instr == "ov1") {
    int pin = input.substring(3, 7).toInt();  //Uložit další znaky z instrukce jako číslo pinu
    pinMode(pin, OUTPUT);
    digitalWrite(pin, HIGH);
    Serial.println("Override 1 on " + String(pin)); //Poslat do pc zprávu o provedení
}else if (instr == "ov0") {
    int pin = input.substring(3, 7).toInt();
    pinMode(pin, OUTPUT);
    digitalWrite(pin, LOW);
    Serial.println("Override 0 on " + String(pin));
}
```

Parsing

Příkazy často obsahují více parametrů než jen jedno slovo, například pokud chceme otočit motor, nestačí pouze název příkazu „motor“, ale je potřeba v příkazu zahrnout i číslo, na jakou pozici se chceme otočit. Do skeneru ale přijde pouze jedna zpráva (například „mtr90“) a program tuto zprávu musí rozkouskovat do jednotlivých částí příkazu. Tomuto rozkouskování se říká **parsing** a je to nedílná součást jakéhokoliv programu, který přijímá instrukce ve formě textu.

Pro zjednodušení programu jsem se rozhodl, že první tři znaky příkazu budou vždy pro typ příkazu a zbývající znaky budou případné instrukce. Pokud příkaz žádné další instrukce

nepotřebuje, například zapnutí laseru, pak se na další znaky ani nedívá. Pokud ale potřebuje příkaz více údajů, přečte je z těchto následujících znaků.

Příkazy:

lon: zkratka pro „laser on“ – zapnout laser

lof: „laser off“ – vypnout laser

cpt: zkratka pro „capture“ – zachytit – udělá fotku a pošle ji do počítače

stn: „standby on“ – zapnout standby režim

stf: „standby off“ – vypnout standby režim

mtr [počet kroků]: motor – pootočí motor na zadanou pozici

tst: test – pošle zpět do počítače jednoduchou odpověď, k otestování komunikace

ov0 [pin]: „override 0“ nastaví v libovolném pinu logickou 0.

ov1 [pin]: „override 1“ nastaví v libovolném pinu logickou 1.

více informací o override příkazu se dozvíte v kapitole 4.2.2.

4.2. Práce s daty

Druhá fáze skenu. Na počítači běží skript, který je v podstatě středem celého projektu. Tento skript od Arduina přijímá obrázky nebo zprávy, které dále zpracovává. Dále do mikropočítače uvnitř skeneru posílá instrukce a třetí fáze tohoto projektu, vykreslování bodů, pracuje se soubory vytvořenými právě tímto skriptem.

4.2.1. Přenos dat do počítače

Po připojení USB kabelu je Arduino v počítači rozpoznáno jako „COM“ port. Tento port lze otevřít v programech jako Arduino IDE pomocí funkce Serial Monitor nebo komunikovat s ním přes jiný software, například mnou napsaný program v jazyce Python.

Vždy když chce nějaký program na počítači získat přístup k tomuto portu, je potřeba vytvořit spojení. Toto spojení lze otevřít pouze, pokud port už nevyužívá žádný jiný program. Když jsem vytvářel program pro počítač pro zpracování dat, musel jsem s touto skutečností počítat. Pokud se v tomto programu pokusíte otevřít komunikaci s Arduinem a nějaký jiný program už ho využívá, můj program to zachytí a upozorní uživatele, že spojení nelze otevřít a je potřeba nejprve uzavřít ostatní programy, které stojí v cestě.

Sériová komunikace používá asynchronní přenos dat, což znamená, že nevyužívá hodinový signál (na rozdíl od SPI nebo I2C). Místo toho se vysílač a přijímač musejí shodnout na několika parametrech:

Baud rate (přenosová rychlost) – např. 9600 nebo 115200 baud – bitů za sekundu

Počet datových bitů – obvykle 8 bitů (možné i 5, 6, 7)

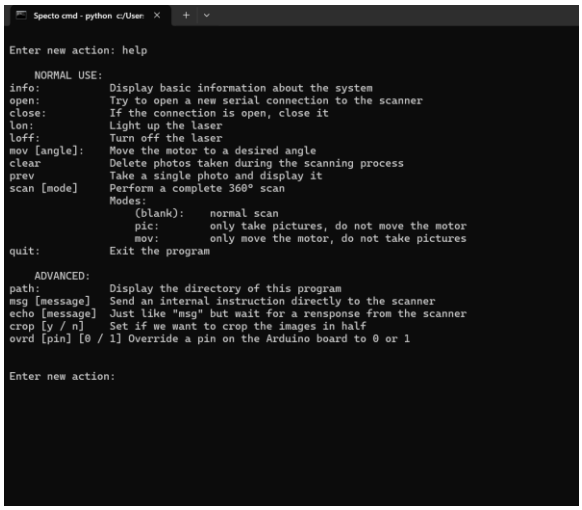
Při přenosu se každý znak odesílá jako samostatný rámec:

| Start bit (0) | 8 datových bitů | Paritní bit (volitelné) | Stop bit (1) |

Paritní bit je přidaný bit k datům pro detekci chyb. Zaznamenává, jestli počet jedniček je sudý (sudá parita) nebo lichý (lichá parita). Pokud se parita jedniček neshoduje s paritním bitem, pak víme že došlo k chybě přenosu a je potřeba data odeslat znovu.

Pokud se vysílač a přijímač neshodnou na parametrech (např. špatný baud rate), data budou nesprávně interpretována.

4.2.2. Ovládání programu



Obrázek 16: Konzole – odpověď programu na příkaz "help"

K ovládání programu jsem zvolil nejjednodušší a nejrychlejší způsob, a to je ovládání přes konzoli.

Příkazová konzole je jednoduché okno, které zobrazuje pouze text a slouží jako prostředí, kterým komunikujete s programem. Funguje v podstatě jako konverzace. Vy napíšete do konzole příkaz, například „help“, a stisknete klávesu **enter.** Tím se příkaz pošle do programu, program příkaz zpracuje a odešle zpět nějakou zprávu, která se opět zobrazí v konzoli.

Pokud nejste absolutní začátečník, programuje se zásadně v angličtině. To je jeden z důvodů, proč jsem zvolil komunikaci uvnitř konzole a zadávání příkazů v anglickém jazyce. Mezi další důvody patří ten, že je většina slov kratší a jednodušší, navíc neobsahuje speciální znaky, například čárky a háčky nad písmeny, které by každé programovací prostředí nemuselo podporovat. Kromě toho je pak jednodušší produkt zpřístupnit ostatním v zahraničí.

Použití probíhá takto: na ploše počítače kliknete na ikonu, tím se spustí program a poté se zobrazí konzole. Program napíše „Enter new action:“ – zadejte novou akci. Uživatel napíše příkaz, třeba „test“, program příkaz zpracuje, v tomto případě otestuje spojení mezi počítačem a skenerem. Potom v konzoli vypíše zpětnou vazbu k tomu, co se událo. Například ohlásí, jestli bylo spojení úspěšné nebo ne. V přiloženém obrázku můžete vidět odpověď programu na příkaz „help“, který vypíše všechny příkazy a jejich užití. Tady si je vysvětlíme v češtině:

Běžné použití:

info: Zobrazí základní informace o programu.

open: Pokusí se otevřít komunikaci se skenerem, aby šel později ovládat.

close: Pokud je otevřená komunikace, ukončit ji. Toto je potřeba, protože komunikaci může využívat jen jeden.

lon: Zapnout laser.

loff: Vypnout laser.

mov [úhel]: Otočí motor na zadaný úhel.

clear: Smaže snímky, které byly pořízeny během skenování.

prev: Pořídí jeden snímek a zobrazí ho na obrazovce počítače. Vhodné pro testování například jestli je kamera zaostřená.

scan [režim]: Provede kompletní 360° sken.

režimy:

(prázdný): Provede celý proces skenu se vším všudy.

pic: Pouze pořídí snímky, motor a tím pádem objekt se nebude pohybovat.

mov: Pouze pohybuje s objektem, nebude pořizovat snímky.

Tyto speciální režimy slouží pouze k testování, ovšem jsou jednoduché k pochopení a jejich použití nevyžaduje žádné zvláštní vědomosti, proto jsem je zařadil mezi běžné použití.

quit: Vypne program.

Pokročilé:

path: Zobrazí cestu na disku, kde se program a související soubory, například snímky, nachází.

msg [zpráva]: Pošle interní příkaz do skeneru. Viz *4.1.4.*

echo [zpráva]: Stejně jako „msg“, ale počká na odpověď od skeneru a zobrazí ji.

crop [y / n]: Nastaví, zda chceme snímky oříznout tak, aby zbyla pouze pravá polovina. Toto je užitečné, pokud je za objektem stěna, na kterou se promítá laserová čára. Naštěstí z geometrického hlediska je tato stěna vždy za středem otočné plochy a tím pádem bude nežádoucí laserová projekce vždy na levé straně snímku. Z toho stejného důvodu bude vždy část, které chceme zachytit na pravé straně. Takže si můžeme být jistí, že při tomto oříznutí se neztrácí žádná data. Zároveň ale není ideální snímky ořezávat vždy, protože kamera neumí zachytit jen část snímku, a tak je potřeba je ořezávat později v programu, což může zbytečně prodloužit celý proces.

ovrd [pin] [0 / 1] Přinutit výstup na jakémkoliv pinu na logickou 1 nebo 0. Toto je velice riskantní příkaz, protože může způsobit problémy, například v komunikaci mezi Arduinem a kamerou. je potřeba provést restart Arduina, aby bylo vše uvedeno do původního stavu.

4.2.3. Extrakce bodů ze snímků

Obrázek 17: Snímek pořízený kamerou, detekovaný obrys – fialová

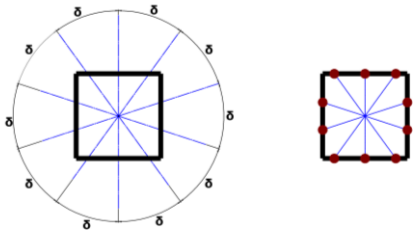
Tuto část bych nazval v podstatě srdcem celého projektu. Mnou definovaná funkce „processImg“ (zpracuj obrázek) načte pořízený snímek, rozpozná laserový obrys objektu a převede ho na soubor souřadnic. Než se pustíme do samotného algoritmu, který rozpozná obrys objektu, je potřeba pochopit, jak počítače vnímají barvy.

Lidské oko má zevnitř sítnici pokrytou třemi typy tyčinek, které reagují na různou vlnovou délku světla – červená, zelená, modrá. Poměrem těchto základních barev pak v našem mozku vznikají další barvy a odstíny. Kamery fungují velice podobně. Uvnitř světlo naráží na světlocitlivou plochu snímače, který obraz převádí na elektrické signály, opět ve známých třech základních barvách. Barvy jdou tedy definovat třemi hodnotami. Tyto hodnoty jsou většinou reprezentovány osmi-bitovými čísly, tedy od 0 do 255. To znamená že bílou popíšeme čísly „255, 255, 255“. Černou „0, 0, 0“, protože černá je vlastně absence světla. Modrou „0, 0, 255“, a podobně.

Jelikož jsou pro zjednodušení tohoto projektu skenované objekty pouze matné a černé, jejich barva by teoreticky (v ideálním případě) měla být samé nuly, a protože laser je červený, pixely, ve kterých se laser nachází by měly mít hodnotu „255, 0, 0“. Samozřejmě nic není perfektní a tyto přesné barvy jsou velmi vzácné. Teorie je ale stále stejná, v tom smyslu že černá (skenovaný objekt) by se měla držet malých čísel, například „20, 20, 20“: odstín šedé. Stejně tak laser by se měl pohybovat kolem vysokého prvního čísla, které reprezentuje červenou a ostatní barvy opět nízko. Například „230, 15, 10“ Hodnoty, které reprezentují jednotlivé složky červené, zelené a modré budu nazývat **r, g, b.** Např. pixel = „230, 15, 10“, pak **r** = 230, **g** = 15, **b** = 10.

S tímto principem jsem vytvořil následující proměnné: koeficient černé („bCoef“) a koeficient červené („rCoef“). Tyto hodnoty reprezentují, jak moc je pixel černý nebo červený a jsou vypočítány jednoduchým vztahem: $\boldsymbol{bkCoef} = \frac{\boldsymbol{r+g+b}}{\mathbf{3}}$, $\boldsymbol{rCoef} = \mathbf{r} - \frac{\boldsymbol{g+b}}{\mathbf{2}}$.

Algoritmus funguje následovně: pro každý řádek pixelů v obrázku, postupně projdeme každý pixel zleva doprava. Zjistíme jeho koeficient černé a také koeficient červené předchozího



Obrázek 18: Pohled seshora: ***skenovaný objekt*** *– černá,* ***jednotlivé snímky*** *– modrá,* ***výsledné body*** *– červená*

pixelu. Tyto koeficienty porovnáme. Pokud je koeficient červené předchozího pixelu nad určitou hranicí, a koeficient černé aktuálního pixelu také nad určitou hranicí, pak to znamená že jsme přešli z červené na černou, neboli předchozí pixel byl součástí obrysu vyznačeného laserem. Tento proces opakujeme pro každý řádek shora dolů.

Pro každý obrázek si vytvoříme seznam, kam budeme ukládat pozice těchto pixelů. Pokud na celém řádku nenajdeme dvojici pixelů, které nevyhovují těmto kritériím, znamená to že nebyl nalezen obrys objektu. V tom případě zapíšeme pozici pixelu **0**. Body na pozici **0** budou později vyfiltrovány. Pokud jsme v řádku našli obrys, zapíšeme do seznamu pozici pixelu na vodorovné ose.

5x5 obrázek	Zapsaná hodnota
	0
	1
	2
	0
	2

Obrázek 19: Jednoduchý příklad dat z obrázku 5 x 5 pixelů

Jelikož víme, kolik snímků bylo pořízeno a víme že snímky jsou rovnoměrně rozprostřené po kružnici, pak podle pořadí snímku jednoduše zjistíme, pod jakým úhlem jsme z aktuálního snímku získávali souřadnice. Úhel lze vypočítat jednoduše: $\delta = \frac{360}{n} * i$ kde **n** je celkový počet snímků a **i** je číslo aktuálního snímku.
Jelikož zapisujeme do seznamu pozice pro všechny řádky, je zbytečné psát do pozice bodů souřadnici ve svislé ose, protože svislou pozici bodu poznáme podle toho, v jakém pořadí je zapsaný. Stačí nám tedy zapsat první číslo v seznamu jako úhel snímku a poté všechny vodorovné pozice. Seznam z jednoho snímku pak může vypadat takto: [δ, x_0, x_1, x_2, ...] například [66.6, 1, 1, 2, 0, 3].

4.2.4. Export zpracovaných dat

Tento proces se opakuje pro všechny snímky pořízené během skenu. Po každém zpracovaném obrázku seznam zapíšeme do textového souboru. Vznikne nám textový soubor, který obsahuje seznam seznamů, neboli 2D seznam. O nich více v kapitole 4.3.1. – Parsing textu.

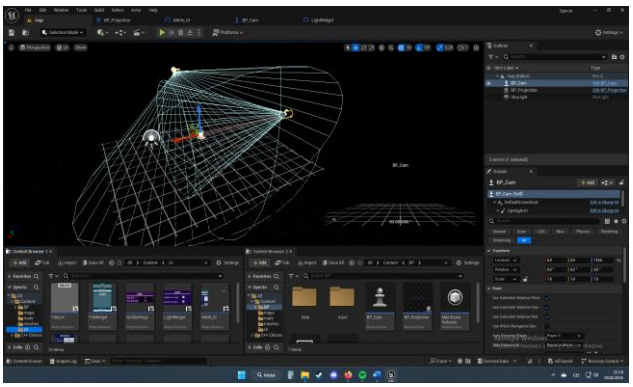
4.3. Zobrazení dat

Třetí a finální fáze skenu: vizualizace získaných dat. Jak bylo zmíněno na začátku cílem je zobrazit získané body ve 3D prostoru. S tím se pojí i další funkce pro zlepšení uživatelské zkušenosti, například otáčení kamery, nebo možnost přepínat mezi různými způsoby vykreslování. Kromě toho je také potřeba vytvořit **uživatelské rozhraní (UI).** Tím se myslí tlačítka, kterými můžete program ovládat. Také textové pole, kam uživatel vloží cestu na disku k souboru, který uchovává data.

4.3.1. Unreal Engine

Vykreslit cokoliv na obrazovku bez pomoci jiných programů je velice náročná záležitost, která vyžaduje rozsáhlé vědomosti programování. Proto jsem se rozhodl, že k vykreslení použiji program, který tento proces velmi zjednodušuje.

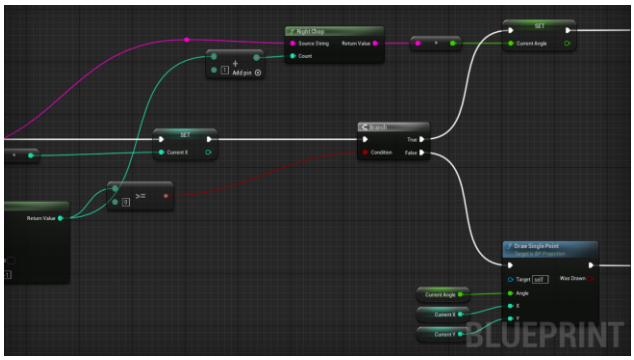
Unreal Engine (UE) je herní engine. V kontextu vývoje her a aplikací je engine platforma, která poskytuje nástroje a prostředí pro tvorbu interaktivních aplikací, nejčastěji videoher. Umožňuje vývojářům soustředit se na tvorbu obsahu, aniž by museli programovat všechny základní funkce od nuly.



Obrázek 20:Prostředí UE

UE je profesionální herní engine vyvinutý společností Epic Games, který slouží k tvorbě videoher, filmových vizualizací, architektonických návrhů, simulací a dalších interaktivních 3D aplikací. Díky své flexibilitě, výkonnému vykreslovacímu systému a pokročilým nástrojům se stal jedním z nejpopulárnějších enginů na světě.

4.3.1.1. Blueprint systém



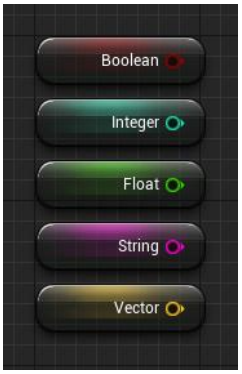
Obrázek 21: Blueprint neboli vizuální skriptování pomocí spojování "uzlů"

V tomto enginu se dá programovat pomocí textu v jazyce **C++** anebo pomocí vizuálního systému tzv. **blueprint**. Tento systém umožňuje vytvářet jakoukoli logiku bez nutnosti textového programování, kde místo psaní kódu propojujete předem připravené bloky. Tento systém je perfektní pro začátečníky ale umožňuje i většinu pokročilých funkcí. Funguje na principu **uzlů (nodes)**, které se propojují **spojnicemi (wires)**, které určují tok provádění kódu. Spojnice mají různé druhy. Tučná bílá určuje směr toku kódu. Ostatní spojnice nesou proměnné a jejich typ poznáme podle barvy. Uprostřed je šedý uzel **branch**, který dělí tok kódu na dvě cesty. Uzly se dají rozdělit do několika kategorií:

Proměnné (Variables)

Proměnné v Blueprintech slouží k ukládání a manipulaci s daty během běhu hry. Jsou základním stavebním kamenem pro vytváření interaktivní logiky a umožňují pracovat s hodnotami, jako jsou čísla, texty, objekty nebo vektory. Každá proměnná má svůj datový typ,

který určuje, jaký druh informace může obsahovat. V UE existuje několik základních typů proměnných:



Obrázek 22: Uzly nesoucí proměnné různých typů

Boolean (True/False) – Uchovává pravdivostní hodnotu (ano/ne). Používá se například pro zapnutí/vypnutí světla nebo aktivaci dveří.

Integer (Celé číslo) – Slouží k ukládání celých čísel, například počtu životů hráče nebo skóre.

Float (Desetinné číslo) – Používá se pro hodnoty s desetinnou čárkou, například rychlost pohybu nebo časovače.

String (Textový řetězec) – Obsahuje textové informace, například jméno postavy nebo hlášku v dialogu.

Vector (Vektor ve 3D prostoru) – Uchovává souřadnice v prostoru (X, Y, Z), například pro určování pozice objektů.

Rotator (Úhly rotace) – Definuje natočení objektů v prostoru, například úhel kamery.

Object (Odkaz na jiný objekt) – Slouží k uchování odkazu na jiný objekt uvnitř hry.

Události (Events)

Jsou to spouštěče logiky v blueprintech. Jsou to uzly, které reagují na různé situace ve hře, jako je začátek hry, kolize, kliknutí myší, stisk klávesy a spustí další logiku.

Event Begin Play – Spustí se, když objekt, ke kterému tato node patří, se objeví ve hře.

Event Tick – Běží každý snímek (frame), což je užitečné pro průběžné aktualizace.

Event Destroyed – Spustí se, když je objekt zničen.

Event Key Pressed (E, F, Space, atd.) – Spustí se při stisku určité klávesy.

Event Mouse Clicked – Aktivuje se, když hráč klikne na objekt.

On Component Begin Overlap – Aktivuje se, když se dva objekty překrývají.

On Component Hit – Spustí se při nárazu dvou objektů.

Také lze vytvořit vlastní eventy, které můžeme spustit z jiných částí kódu.

Funkce a Makra

Funkce v UE jsou základním nástrojem pro organizaci a blueprintů. Umožňují uživatelům oddělit často používané části kódu do samostatných bloků (nodes), které lze snadno volat(spustit) z různých míst. To pomáhá udržet přehlednost, zamezit opakování kódu a zefektivnit jeho údržbu.

Každá funkce může mít vstupní a výstupní parametry, což znamená, že může přijímat hodnoty, provádět výpočty nebo jiné operace a následně vrátit výsledek. Například funkce

určená k výpočtu celkového počtu bodů může jako vstup přijímat seznam hodnot, provést výpočet a vrátit výsledek. Díky tomu lze snadno změnit způsob výpočtu pouze úpravou funkce, aniž by bylo nutné měnit kód na několika různých místech.

Jednou z důležitých vlastností funkcí je izolace proměnných. To znamená, že proměnné vytvořené uvnitř funkce nelze získat nebo upravit mimo funkci. Každé volání funkce pracuje se svými vlastními hodnotami, což zabraňuje nechtěným změnám v hlavním blueprintu. Tato izolace je užitečná při výpočtech, kde nechceme, aby dočasné změny ovlivnily globální stav aplikace.

Funkce v blueprintech mají jedno omezení – nemohou obsahovat časové zpoždění (Delay). Všechny operace uvnitř funkce musí být provedeny okamžitě v rámci jednoho snímku. To znamená, že pokud je potřeba čekat určitou dobu před dalším krokem, například při animaci postavy nebo čekání na odpověď od serveru, je nutné použít jiný mechanismus, například dříve zmíněné Eventy nebo Timery (časovače).

Další výhodou funkcí je jejich přehlednost a možnost snadného testování. Pokud je například v kódu problém s výpočtem, lze funkci snadno izolovat a otestovat samostatně, což pomáhá rychleji najít chyby.

Funkce jsou tedy klíčovým prvkem při vytváření efektivních a dobře organizovaných blueprintů. Pomáhají snižovat složitost kódu, zvyšují jeho opakovatelnost a umožňují snadnou údržbu. Při jejich správném používání je možné vytvářet robustní a škálovatelné herní systémy, které se snadno rozšiřují a upravují podle potřeb projektu.

Podmínky (Branch)

Podmínky v blueprintech umožňují rozvětvení kódu na dvě větve. Podle toho, která větev bude spuštěna rozhoduje vstupní binární proměnná (Boolean). Jedná se o ekvivalent operátora **IF** v textovém programování. Složitější logiku lze tvořit pomocí operátorů **AND**, **OR** a **NOT**, které kombinují více podmínek.

Smyčky (Loops)

Smyčky v blueprintech umožňují opakované provádění stejné logiky bez nutnosti manuálního opakování kódu. Nejčastěji používané jsou **For Loop**, **For Each Loop** a **While Loop**. **For Loop** probíhá od čísla A, a opakuje danou logiku, dokud nedosáhne čísla B. **For Each Loop** slouží k procházení všech prvků v seznamu, tuto smyčku jsem použil při vykreslování jednotlivých bodů v seznamu snímku. **While Loop** běží tak dlouho, dokud je splněna podmínka, což se používá hlavně ve hrách a zde není moc relevantní.

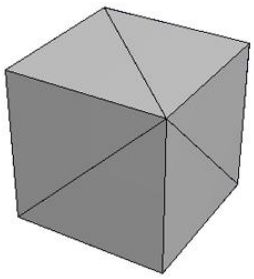
4.3.1.2. Rendering

Takzvaný **Rendering pipeline** je proces, kterým se 3D scéna převádí na finální 2D obraz zobrazený na obrazovce. Tento proces probíhá v několika fázích a je optimalizován pro rychlost, zejména v aplikacích, které běží v reálném čase, jako jsou videohry nebo jiné programy.

Celý rendering začíná v procesoru, kde běží hlavní logika aplikace. Herní engine připraví data, jako jsou 3D modely, pozice, světla, animace a pozice kamery. Tato data jsou poté odeslána do grafické karty přes grafické API (Application Programming Interface), například DirectX, OpenGL nebo Vulkan, kde probíhá samotné vykreslování.

První fáze renderování je geometrická fáze, kde se jednotlivé body(vertexy) modelů transformují z jejich lokálních souřadnic do obrazového prostoru kamery. Tato transformace zahrnuje změny perspektivy, umístění objektů ve scéně a ořezávání částí, které se nacházejí mimo zorné pole kamery. V této fázi pracuje **vertex shader**, který upravuje polohu a další vlastnosti vertexů, jako je osvětlení nebo barva.

Dalším krokem je **rasterizace**, kde se trojúhelníky, ze kterých se skládají modely, převedou na pixely na obrazovce. Tento proces zahrnuje rozdělení modelů na jednotlivé fragmenty (pixely), přičemž **fragment shader** určuje jejich barvu podle textur, světel a dalších efektů, jako jsou odrazy nebo průhlednost. V této fázi se také provádí **hloubkový test (Z-buffer)**, který zajišťuje, že se vykreslují pouze viditelné části objektů, zatímco ty, které jsou zakryté jinými, jsou ignorovány.



Obrázek 23: I obyčejná krychle se ve skutečnosti skládá z trojúhelníků.

Poslední fáze je **post-processing**, kde se na obraz aplikují různé vizuální efekty, ty ale nejsou důležité. Celý rendering pipeline je optimalizován tak, aby probíhal co nejrychleji, což je klíčové pro plynulý běh her a interaktivních aplikací.

5. Výsledný sken

Nejprve bych vám rád popsal, jak tedy funguje aplikace pro zobrazení skenů.

5.1. Uživatelské rozhraní

Uživatel do textového pole zadá lokaci souboru, který vytvořil program napsaný v Pythonu z předchozí kapitoly. Potom klikne na tlačítko, které předá pokyn k vykreslení souboru.

5.2. Přečtení souboru

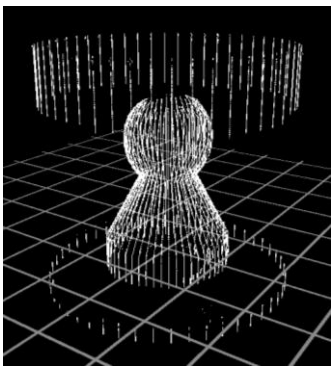
Spustí se funkce, která přečte zadaný soubor. Tato část programu je jediná, která nešla udělat pomocí vizuálního skriptování, a tak jsem ji musel napsat v C++. Tato funkce soubor přečte a text uloží jako proměnnou, se kterou už můžu pracovat pomocí vizuálního programování.

5.3. Parsing textu

Jak jsem již vysvětlil v kapitole 4.1.4, parsing je proces, kdy souvislý text rozdělíme na více různých proměnných nebo instrukcí. Uvnitř UE je sken reprezentovaný tzv. dvoudimenzionálním seznamem. To znamená že je to seznam několika seznamů. Konkrétně každý **snímek** je seznam bodů a sken je **seznam snímků**. My použijeme hranaté závorky, abychom rozdělili text do jednotlivých snímků. Potom víme, že vždy první číslo v seznamu jednoho snímku je úhel, pod kterým byl snímek pořízen. Další čísla, rozdělená čárkou, jsou jednotlivé body.

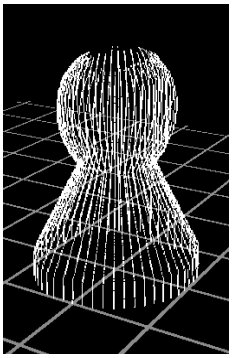
5.4. Filtrování

V kapitole 4.2.3. jsem zmínil, že body na pozici 0 budou vyfiltrovány. Během vykreslování probíhá poslední fáze práce s daty a to je odstranění nežádoucích bodů. Tím se myslí odstranit všechny body na pozici 0 ale také všechny body pod určitou hloubkou. Tyto body jsou totiž naskenovaná podložka a nepatří do našeho objektu.



Obrázek 24: Zobrazení všech bodů bez filtrování

5.5. Vykreslení

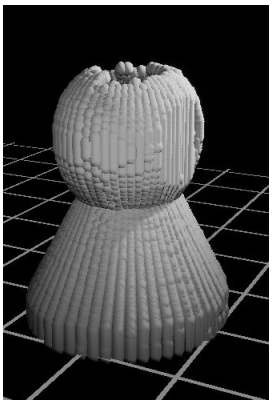


Obrázek 25: Bodové zobrazení skenu s filtrováním

Aktuálně jsou dostupné dva způsoby vykreslování, které jsem naprogramoval. První je vykreslování čistých bodů. Tyto body jsou jedna solidní barva, nijak nereagují na osvětlení ani vzdálenost od kamery. Jejich velikost je zadána v pixelech na obrazovce. To znamená, že nezávisle na tom, jak je kamera blízko u bodu, vždy bude mít stejnou velikost. Kvůli těmto limitacím je v některých případech obtížné rozpoznat naskenovaný tvar. Na druhou stranu tato metoda je velice výpočetně nenáročná, a proto ji používám častěji.

Druhý způsob je složitější. Ten funguje tak, že na pozici každého bodu vykreslím jednoduchý model (**mesh**) koule. Tento způsob je velice náročný, protože je potřeba vykreslit desetitisíce modelů. Díky tomu ale lze vytvořit relativně dobrou vizualizaci skenu. Koule společně vytvoří iluzi celistvého objektu a tvar skenu bude lépe rozeznatelný. Navíc může reagovat na světlo, stíny a podobně.

Obrázek 27: Testovací kuželka



Obrázek 26: Finální výsledek – objemové zobrazení i s osvětlením

5.5.1. Teoretický limit

V přiložených snímcích můžete vidět sken kuželky z **48 snímků**. Celkový počet získaných bodů je **15 300.** S filtrováním nežádoucích bodů počet klesá na **8 550**. Potenciál skeneru je ale mnohem větší. S použitím nejjemnějších mikrokroků, tedy 32 mikrokroků za jeden krok, motor se může otočit až na 6 400 různých úhlů a tedy tento počet různých snímků. Také jsem používal na kameře nastavení s nejmenším rozlišením 240 x 320 pixelů. Kamera má ale teoreticky nejvyšší rozlišení 2592 x 1944 px. Pokud za jeden snímek zachytíme obrys na dvou třetinách obrazovky, pak se bavíme o 1 600 bodů za snímek. **Teoretické maximum je cca 10 240 000 bodů.**

5.5.2. Optimalizace

Stoprocentně perfektní sken není fyzicky možné provést, ale čím více bodů zachytíme, blíže k přesnosti se dostaneme. Proto je ale potřeba dokázat vykreslit deseti až statisíce bodů. Kvůli tomu je nutné využít co nejvíce metod, jak program zrychlit.

První metody jsou už předem zabudované v enginu, který používám. Například **frustum culling**, což znamená že veškerá geometrie, která je mimo kameru nebude nijak započítávána. Nebo samotná metoda vykreslování bodů, kterou používám. Ta je vyvinutá pro pomocné vykreslování hodnot při tvorbě her. To znamená že tvůrci enginu ji vytvořily za jediným účelem, a to je, aby byla co nejrychlejší. Jakákoliv jiná metoda vykreslování bodů by byla pomalejší.

Další metoda pro optimalizaci je použití „Instanced mesh“ pro vykreslení modelů koulí. Je to technika v počítačové grafice, která umožňuje efektivně vykreslovat mnoho kopií stejného 3D objektu (meshe) s různými transformacemi (pozicí, rotací, měřítkem). Místo toho, aby se každý objekt vytvářel zvlášť, GPU vykreslí jednu instanci modelu a pak ji "naklonuje" s různými parametry. To šetří výkon, protože data modelu jsou uložena jen jednou a změny se aplikují jen na jejich zobrazení.

6. Plány do budoucna

Věřím, že tento projekt má potenciál a chtěl bych na něm pracovat i v budoucnu buď jako bakalářskou práci, nebo jen jako vlastní osobní projekt.

6.1. Více úhlů

Jeden z největších problémů aktuálního řešení spočívá v tom, že pokud je skenovaná stěna kolmá nebo odvrácená od kamery, laserový paprsek se do ní neodrazí a skener tuto část objektu nezachytí. To znamená, že například krychli lze naskenovat pouze z jedné poloviny, tedy přibližně z 50 %, což vede k nekompletnímu modelu. Tento nedostatek může být zvláště problematický u složitějších tvarů, kde některé části objektu zůstávají zcela neviditelné pro skener.

Jedním z možných řešení by bylo přidání druhé kamery, která by byla umístěna symetricky k té první tak, aby laser tvořil mezi oběma kamerami osu souměrnosti. Tím by se pokryly i ty části objektu, které jsou pro jednu kameru neviditelné, čímž by se výrazně zvýšila kvalita a úplnost skenu.

Další možností by bylo namontovat kameru na otočný mechanismus, který by jí umožnil pohyb kolem osy objektu. Tento systém by zajistil snímání objektu z různých úhlů, což by fakticky nahradilo druhou kameru. Díky tomuto řešení by bylo možné získat mnohem detailnější a přesnější sken, aniž by bylo nutné zvyšovat počet kamer a složitost systému. Tento přístup by navíc mohl přispět k větší flexibilitě celého skeneru, protože by umožňoval lepší přizpůsobení různým typům objektů a jejich tvarům.

6.2. Lepší zpracování dat

V programu, který zpracovává fotografie na čistá data, existuje značný prostor pro zlepšení. V současné verzi zatím nezohledňuji prostorové zkreslení, což vede k tomu, že výsledné skeny tvarově neodpovídají realitě. Tento nedostatek může způsobovat nepřesnosti zejména u složitějších objektů, kde je důležité správně interpretovat perspektivu a geometrii.

Dalším problémem je jednoduchost mého algoritmu pro rozpoznávání laserového obrysu. V aktuálním stavu je poměrně primitivní, což v některých světelných podmínkách vede ke vzniku výrazného zrnění a artefaktů. Tyto chyby mohou negativně ovlivnit kvalitu skenu.

Řešení obou problémů je teoreticky jasné – bude nutné experimentovat s různými metodami a postupně zjistit, které přístupy fungují nejlépe. To zahrnuje testování různých algoritmů pro korekci zkreslení, filtrování šumu a optimalizaci rozpoznávání laserového paprsku, což by mělo vést k výrazně kvalitnějším a přesnějším skenům.

6.3. Vykreslování dat

Aktuální program na zobrazení dat už má několik funkcí k úpravě vzhledu, ovládání světel a podobně. Je to ale právě tento program, kde mám ještě velké plány. Mám v plánu přidat možnost, která automaticky rozpozná artefakty skenování, které nepatří do tvaru skenovaného

objektu. Věřím, že to nemusí být tak složité, jak se může prvně zdát. Stačí zachovat pouze body, které na sebe nějak pravidelně navazují, oproti chybným bodům, které se objevují v podstatě náhodně.

Na co se ale nejvíce těším je funkce rozpoznání tvaru. Vím, že to nebude jednoduché, ale zároveň věřím, že na to eventuelně přijdu a těším se na tu výzvu.

6.4. Fyzická skladba

Vzhledem k omezenému rozpočtu jsem se rozhodl objednat pouze jeden univerzální model motoru, který mohu v případě potřeby vyjmout ze skeneru a použít v jiném projektu. Z tohoto důvodu jsem zvolil větší a silnější motor, i když jeho výkon není pro samotný skener nezbytný. Pokud bych nakupoval komponenty výhradně pro skener, upřednostnil bych co nejmenší motor, protože síla pohonu zde nehraje významnou roli.

Do budoucna plánuji volit komponenty s důrazem na co nejmenší rozměry, nejen proto, aby byl samotný skener kompaktní, ale také kvůli lepší organizaci vnitřního prostoru. Čím více volného místa uvnitř zůstane, tím snazší a přehlednější bude propojení jednotlivých částí.

Pokud se rozhodnu pokračovat v projektu v rámci bakalářské práce, pravděpodobně začnu úplně od nuly a celý skener přepracuji tak, aby byl optimalizovaný jak z hlediska konstrukce, tak z pohledu efektivity použitých komponentů.

6.5. Sjednocení programů

Aktuálně je na počítači potřeba nejprve spustit Python program pro ovládání skeneru a zpracování snímků. Poté většinou program zavřu, aby zbytečně nezpomaloval počítač. Potom otevřu druhý program, který mi sken zobrazí.

Prvním krokem ke sjednocení bude místo Unreal Engine přejít na jiný engine, například Godot. UE je totiž primárně tvořená pro náročné videohry a je pro mé účely zbytečně výkonnostně náročná a zabírá moc místa na disku. Godot je mnohem jednodušší a méně komplexní, tím je více vhodný pro mé účely. [6]

Dalším krokem je pomocí této aplikace v Godotu spustit a ovládat python program tak, abych ve výsledku mohl všechno ovládat z jednoho místa.

7. Závěr

Cílem práce bylo naskenovat jednoduchý objekt a sken na počítači zobrazit. Tento cíl byl úspěšně splněn. Elektronické součásti jsem sám vybral a nakoupil přes internet. Zbytek fyzické stránky projektu jsem vytiskl na své vlastní 3D tiskárně. Napsal jsem 3 různé programy pro různé fáze skenování. Napsal jsem vlastní kód do Arduina pro ovládání kamery, laseru a motoru v C++. Na počítači jsem napsal program v Pythonu pro zpracování fotografií a převedení na čistá data. Dále jsem vytvořil program pro zobrazování těchto dat v Unreal Engine pomocí C++ a vizuálního skriptování.

Po splnění minimálního zadání jsem pokračoval dále v projektu. Fyzickou schránku jsem navrhnul v prostředí SolidWorks a zdokonalil jsem ji tak, aby šel skener jednoduše složit a rozložit. Do všech programů jsem přidal více funkcí pro jednoduší použití a testování. Také jsem dále optimalizoval software tak, aby celý proces byl jednodušší a rychlejší. A hlavně, co považuji za nejdůležitější: všechen software jsem vytvořil tak, aby fungoval jako základ, na který můžu později stavět více funkcí.

Tento projekt dopadl skvěle. Rozšířil mi znalosti jak v oblasti elektroniky, 3D tisku, CAD designu ale také programování v C++ a Pythonu. Věřím, že má potenciál a chtěl bych na něm pracovat i v budoucnosti.

8. Reference

[1] „RPiShop," [Online]. Available: https://rpishop.cz/spi-kamerove-moduly/2800-arducam-5mpx-ov5642-spi-mini-kamera-shield.html.

[2] „LaskaKit - Motor," [Online]. Available: https://www.laskakit.cz/krokovy-motor-nema-17-17hs3401-0-28nm/.

[3] „LaskaKit - Driver," [Online]. Available: https://www.laskakit.cz/drv8825-driver-pro-krokove-motory/.

[4] „Botland - Driver," [Online]. Available: https://botland.cz/ovladace-krokovych-motoru/1324-drv8825-ovladac-krokoveho-motoru-45v-22a-pololu-2133-5903351244824.html.

[5] „LaskaKit - Arduino," [Online]. Available: https://www.laskakit.cz/arduino-uno-rev3--original/.

[6] „Godot," [Online]. Available: https://godotengine.org/.

[7] „Unreal engine 5 Documentation," [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation.

9. Seznam obrázků