Markéta Trnečková

OpenSCAD

Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci

Copyright © 2024 Katedra informatiky, Univerzita Palackého v Olomouci
www.inf.upol.cz

Tento text slouží jako učební materiál pro kurz 3D tisk, který je vyučován v bakalářském studiu na Katedře informatiky Přírodovědecké fakulty Univerzity Palackého v Olomouci.

Poslední změna proběhla 29. října 2024. Pokud v textu naleznete chyby, prosím, kontaktujte mne prostřednictvím e-mailové adresy marketa.trneckova@upol.cz.

Děkuji Martinu Trnečkovi za jeho připomínky a cenné podněty, které přispěly ke kvalitě textu.

Markéta Trnečková

Obsah

Úvod **7**
Konstruktivní geometrie . . . 7

OpenSCAD **11**
Prostředí OpenSCAD . . . 11
Program . . . 13

Trojrozměrné objekty **15**
Geometrická primitiva . . . 15
Krychle . . . 15
Válec . . . 16
Koule . . . 18
Mnohostěn . . . 18
Importování objektů . . . 20
Výšková mapa . . . 21

Kombinace trojrozměrných objektů **23**
Transformace . . . 23
Změna velikosti . . . 23
Otočení . . . 24
Posunutí . . . 24
Zrcadlení . . . 25
Afinní transformace . . . 25
Skládání transformací . . . 26
Aplikace transformace na více objektů . . . 26
Operace . . . 26
Další způsoby kombinování objektů . . . 28
Ladění . . . 28
Modifikátor % (background) . . . 28
Modifikátor # (debug) . . . 28
Modifikátor ! (root) . . . 29
Modifikátor * (disable) . . . 29
Změna barvy . . . 29

Dvourozměrné objekty **33**
Dvourozměrné objekty . . . 33
Čtverec/obdélník . . . 33

Kruh . . . 33
Mnohoúhelník . . . 34
Text . . . 34
Import 2D objektů . . . 34
Transformace a množinové operace . . . 34
Offset . . . 35
Vytvoření trojrozměrných objektů z dvourozměrných . . . 37
Lineární extruze . . . 37
Rotační extruze . . . 39
Vytvoření dvourozměrného objektu z trojrozměrného . . . 40

Programování **45**
Hodnoty . . . 45
Čísla . . . 45
Logické hodnoty . . . 46
Řetězce . . . 46
Vektory . . . 47
Rozsah . . . 48
Funkce pro ověřování datových typů . . . 49
Proměnné . . . 49
Platnost proměnných . . . 50
Cykly . . . 51
Podmínky . . . 53

Funkce a moduly **57**
Funkce . . . 57
Rekurzivní funkce . . . 59
Anonymní funkce . . . 59
Moduly . . . 60
Objektové moduly . . . 60
Moduly představující operace . . . 61
Rekurzivní moduly . . . 62
Funkce `parent_module()` . . . 62
Knihovny . . . 63

Co se jinam nevešlo **67**
Modul `render()` . . . 67
Speciální proměnné . . . 67
Animace: proměnná `$t` . . . 68
Viewport: `$vpr`, `$vpt`, `$vpf` a `$vpd` . . . 69
Preview: `$preview` . . . 69
Aserce . . . 70
Externí editory . . . 70

Řešení vybraných úkolů **73**

Úvod

Ve světě 3D modelování existuje bohatý výběr softwarových nástrojů, které slouží k vytváření trojrozměrných objektů. Ty se liší způsoby, jakými je model vytvářen. Objekty je možné tvořit v různých *CAD software*[1] jako jsou například TinkerCAD, FreeCAD nebo OpenSCAD,[2] kterému se budeme věnovat. CAD je založený na konstruktivní geometrii, kde je objekt reprezentován popisem konstrukce z jednoduchých těles, kterým říkáme geometrická primitiva.[3] [5] Modelům, které je možné popsat pomocí geometrických primitiv, se také říká *hard solid*. V CAD však není možné vymodelovat objekty, které takto popsat nejde, například obličej nebo postavu. Takové objekty je potřeba modelovat pomocí modelovacích programů určených pro umělecké modelovaní, jakým je například Blender.[4] Zde je objekt reprezentován jako síť bodů a ploch (mesh), která se pomocí různých nástrojů upravuje.
V tomto textu se budeme věnovat modelování pomocí OpenSCAD. Ten se od ostatních CAD výrazně odlišuje. V OpenSCAD se modeluje pomocí jednoduchého programovacího jazyka (jenž se nazývá také OpenSCAD), kterým popisujeme postup konstrukce výsledného objektu. Díky tomu má designér větší kontrolu nad modelovaným objektem a umožňuje mu to kdykoliv v průběhu jednoduše změnit jakoukoliv část návrhu. Zejména proto je OpenSCAD oblíbenou volbou pro návrh 3D objektů určených k 3D tisku.

[1] CAD = Computer Aided Design, česky počítačem podporované projektování.

[2] Čteme jako Open-S-CAD.

[3] Geometrickými primitivy mohou být například koule, krychle, ale i různé plochy.

[4] `https://www.blender.org/`

Konstruktivní geometrie

Jednou z reprezentací 3D objektů v počítači je pomocí *konstruktivní geometrie těles*. V té je objekt popsán způsobem, který odráží postup, jaký by konstruktér použil při jeho návrhu. Z jednoduchých geometrických tvarů, kterým říkáme *geometrická primitiva*, je pomocí *množinových operací* a *prostorových transformací* vytvořen výsledný objekt.
Mezi geometrická primitiva patří základní geometrické tvary, například krychle, koule, kužel nebo válec. Množinové operace jsou sjednocení, rozdíl a průnik. *Sjednocením* dvou množin získáme množinu, která obsahuje všechny prvky z obou množin. *Rozdílem* dvou

Průvodce studiem
Constructive solid geometry (CSG, konstruktivní geometrie těles) je vektorové modelování geometrických objektů. Tyto objekty se konstruují z primitivních geometrických těles (například koule, kvádr, válec, kužel) za pomoci množinových operací (sjednocení, průniku a rozdílu).

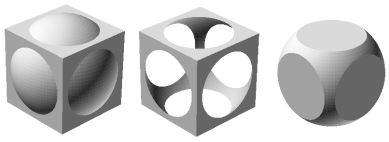
množin rozumíme všechny prvky, které patří do první množiny, ale nepatří do druhé. *Průnik* dvou množin je množina společných prvků, tedy prvků, které patří do obou množin. Na obrázku 1 jsou po řadě výsledky sjednocení, rozdílu a průniku krychle a koule. Příkladem prostorových transformací jsou posunutí v nějakém směru, otočení okolo některé z os v prostoru nebo změna velikosti. Velikost objektů lze změnit jejich prodloužením nebo zkrácením v jednom směru nebo jejich proporcionálním zvětšením či zmenšením.

Metoda konstruktivní geometrie těles, anglicky constructive solid geometry, zkráceně CSG, je založena na reprezentaci tělesa stromovou strukturou tzv. *CSG stromem*, uchovávající dílčí kroky konstrukce.

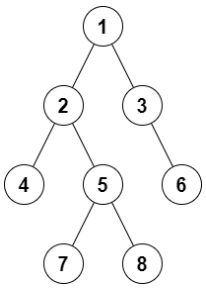
Stromovou strukturu si lze zjednodušeně představit jako uspořádanou množinu prvků, kterým říkáme *uzly*. Mezi těmito uzly existuje hierarchický vztah potomek-rodič. Říkáme, že rodič je v hierarchii (uspořádání) nad potomkem (případně potomek je v hierarchii pod rodičem). Pokud uzel nemá žádného rodiče, nazývá se *kořenem stromu*. V každém stromu existuje právě jeden kořen. Prvky, které nemají žádného potomka, se nazývají *listy stromu*. Obrázek 2 znázorňuje příklad stromu. Uzel označený číslem 1 je kořenem stromu, uzly 4, 6, 7 a 8 jsou listy stromu. Uzel 1 je rodičem uzlů 2 a 3. Uzel 2 má dva potomky a to uzly 4 a 5. Listy CSG stromu představují geometrická primitiva, ostatní uzly představují některou z operací nebo transformaci, která je aplikována na potomky uzlu. Těleso popsané CSG stromem je postupně konstruováno od listů směrem ke kořenu. Příklad CSG stromu je znázorněn na obrázku 2. V tomto příkladu začínáme s torem,[5] krychlí a dvěma válci, jedním větším a jedním menším. Od toru odečteme krychli (operace je označena -) a od většího válce odečteme menší. Výsledky těchto operací můžete vidět vedle uzlů. Tyto výsledné objekty sjednotíme (operace $\cup$). Výsledný objekt leží v kořenu stromu. Pro úplnost dodejme, že operace průniku, která se v našem příkladu nevyskytuje, se značí symbolem $\cap$.

Pro jednoduchost jsou ve stromu na obrázku 3 vynechány prostorové transformace. Například jsme mohli začínat se dvěma stejnými válci a jeden z nich zmenšit. Dále v příkladu předpokládáme správné umístění toru vzhledem ke krychli, kterou od něj odečítáme.

Obrázek 1: Sjednocení, rozdíl a průnik krychle a koule.

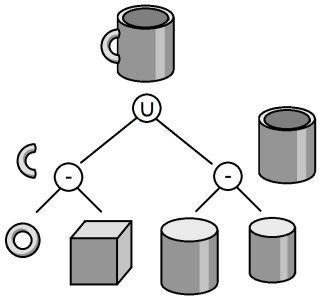


Obrázek 2: Strom.



[5] Torus je těleso, které vznikne otáčením kružnice kolem osy, která leží ve stejné rovině a nemá s ní společné body.

Obrázek 3: CSG strom.



Shrnutí

V této kapitole jsme uvedli několik příkladů modelovacích nástrojů. Vysvětlili si pojem konstruktivní geometrie těles, jejíž princip se používá při modelování v CAD softwarech.

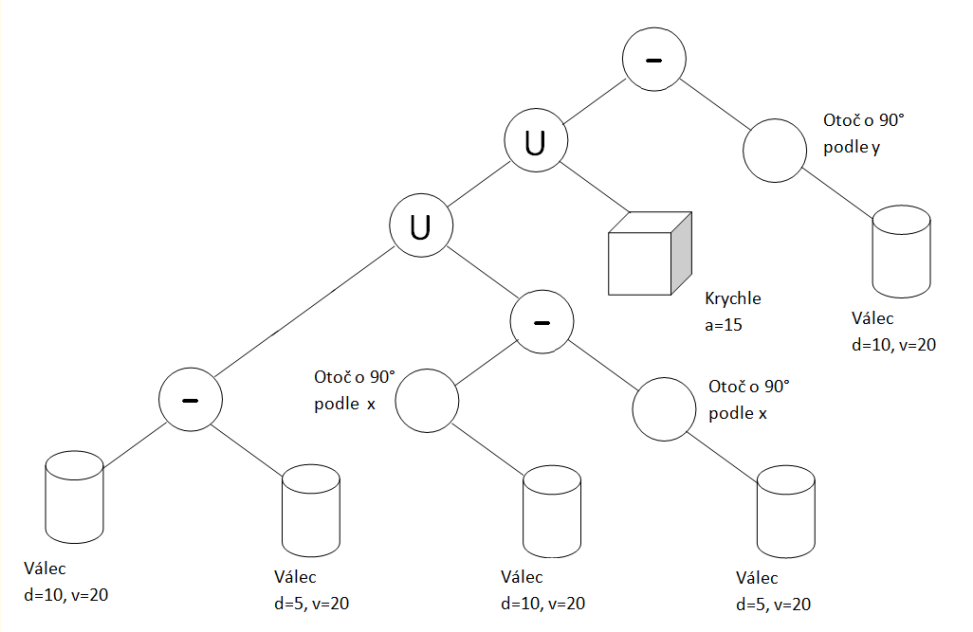
Kontrolní otázky
Odpovězte na následující otázky:

1. Co je to konstruktivní geometrie?
2. Uveďte příklad geometrických primitiv.

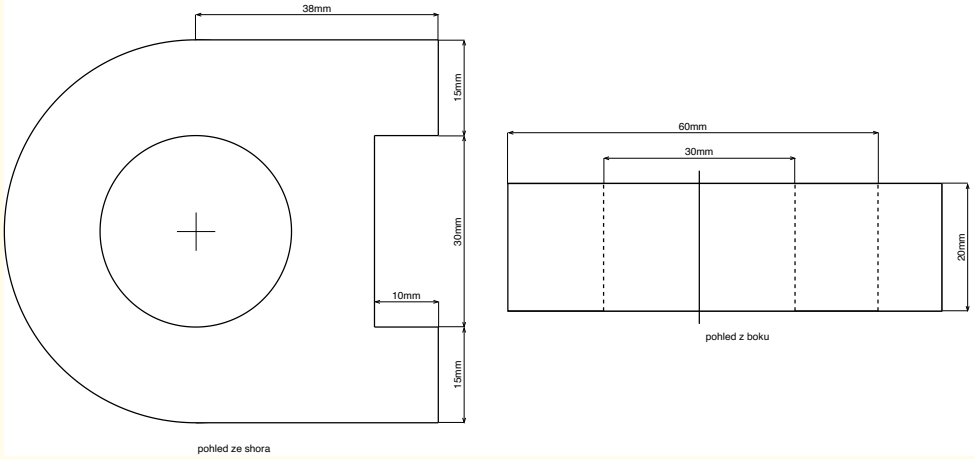
Cvičení

Úkol 1
Popište jak pomocí sjednocení a rozdílu těles udělat jejich průnik.

Úkol 2
Nakreslete výsledné těleso, které je reprezentované CSG stromem na obrázku níže. Středy jednotlivých těles jsou ve stejném bodě. Otáčení tělesa je podle jeho středu.
d označuje průměr, v výšku, a délku strany krychle a x, y, z představují jednotlivé osy prostoru. Otočení podle osy x tedy znamená otočení ve směru zepředu-dozadu, a kolem osy y zleva-doprava.



Úkol 3
Nakreslete CSG strom následujícího tělesa.



OpenSCAD

OpenSCAD je nástroj pro tvorbu 3D modelů. Je zdarma dostupný na adrese

`https://www.openscad.org`

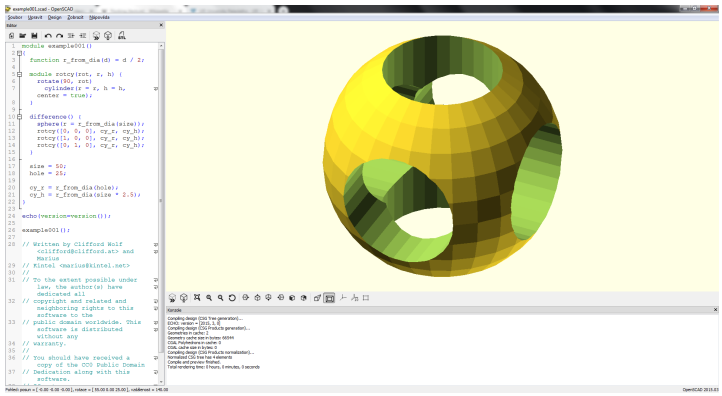
ve verzích pro operační systémy MS Windows, Linux a macOS. OpenSCAD slouží jako *překladač* (kompilátor), který čte textový popis tělesa a podle něj těleso vykresluje.
OpenSCAD je deklarativní programovací jazyk.[6] Jeho dokumentaci nalezneme v [1]. OpenSCAD poskytuje dva základní způsoby tvorby 3D objektů. Jedním ze způsobů je již dříve zmíněná konstruktivní geometrie, druhým je extruze[7] 2D tvarů do prostoru. 2D tvary opět můžeme popsat pomocí konstruktivní geometrie s tím, že v tomto případě jsou geometrickými primitivy základní geometrické tvary, jako například obdélníky, kružnice nebo mnohoúhelníky.

[6] V jazyce popisujeme tvar, velikost a způsob kombinace jednotlivých částí, ne to, jak konkrétně toho dosáhneme.

[7] Extruze je proces, při kterém se tvaruje materiál tím, že se vytlačuje skrz otvor, čímž vytváří objekty s konstantním průřezem.

Prostředí OpenSCAD

Než začneme tvořit, seznámíme se s prostředím programu OpenSCAD, které je zobrazeno na obrázku 4.

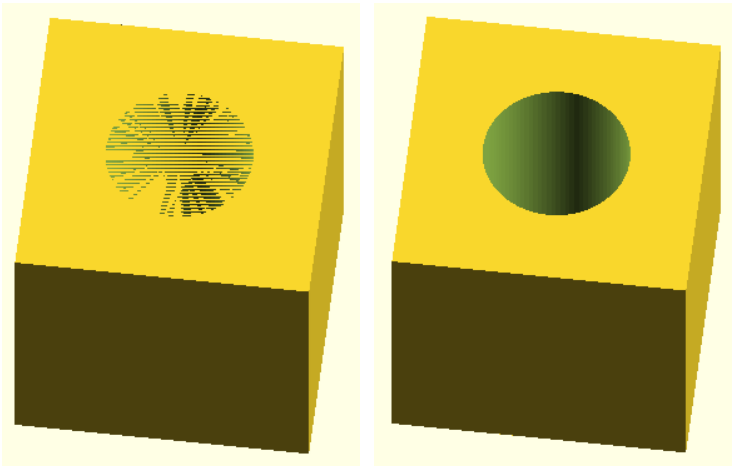


Obrázek 4: Vývojové prostředí.

Průvodce studiem
OpenSCAD je software určený k vytváření 3D modelů. Na rozdíl od jiných programů podobného zaměření se v něm modely vytváří textovým popisem pomocí speciálního procedurálního programovacího jazyka.

V levé části okna se nachází *editor*, kde se zapisuje kód popisující těleso. Nad editorem se nachází několik tlačítek. Kromě tlačítek pro načtení, uložení, úpravy odsazení kroku vpřed a zpět, zde najdeme tlačítka `Zobrazit`[8] a `Vyrenderovat`[9]. Obě slouží k přeložení kódu a vykreslení popsaného tělesa. První je rychlejší a vykreslí jen náhled výsledného tělesa, který nemusí vždy přesný. Druhé tlačítko slouží k plnohodnotnému vyrenderování objektu. Na obrázku 5 můžeme vidět rozdíl mezi výsledky pro těleso, které je popsáno rozdílem krychle a válce.

[8] Klávesová zkratka `F5`.
[9] Klávesová zkratka `F6`



(a) Zobrazit (b) Vyrenderovat

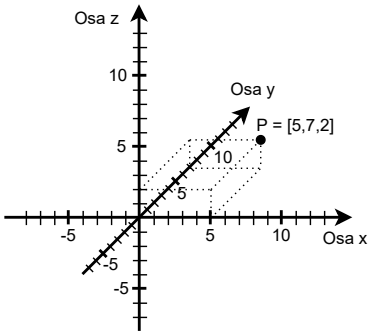
Obrázek 5: Překlad kódu.

Pravá část okna je rozdělena na dvě. Horní část, představující virtuální trojrozměrný prostor, je *náhledová oblast* (viewing area), ve které se po přeložení kódu zobrazí modelované těleso. Body tohoto prostoru jsou zadávány v kartézské soustavě souřadnic. Kartézská soustava souřadnic je soustava souřadnic, u které jsou souřadné osy vzájemně kolmé přímky, které se protínají v jednom bodě – počátku soustavy souřadnic. Na obrázku 6 máme znázorněn bod P v kartézské soustavě souřadnic. Tento bod má souřadnice $[5,7,2]$. Bod na souřadnicích $[0,0,0]$ nazýváme *počátek souřadnic*.

Pracovní plochu je možné za pomoci myši posouvat,[10] otáčet[11] a přibližovat/oddalovat ji.[12] V menu je možné si plochu přizpůsobit, například je možné si zapnout nebo vypnout zobrazení os a pravítka. Část pod náhledovou oblastí slouží pro výpis zpráv týkajících se překladu, chyb a varování, případně uživatelských výpisů.

K výpisu do konzole slouží příkaz `echo()`, kde se do závorek napíše výraz (textový řetězec, konstanta, proměnná,...), který se má vypsat.

Obrázek 6: Kartézská soustava souřadnic.



[10] Při stisku pravého tlačítka myši a jejím tažením.

[11] Při stisku levého tlačítka myši a jejím tažením.

[12] Pomocí kolečka myši.

Program

Program zapsaný v programovacím jazyce OpenSCAD představuje popis nějakého modelu. Základními stavebními kameny téměř každého programovacího jazyka jsou *hodnoty*, *výrazy* a *příkazy*. Hodnoty reprezentují informace (data), se kterými programy pracují. Například to mohou být čísla, textové řetězce a jiné. K jejich vytváření slouží výrazy. Příkazy dávají počítači instrukce, co má udělat, například vytvořit geometrické primitivum. Jednotlivé příkazy se v kódu oddělují středníkem (znakem `;`).
Nejprve si ukážeme pouze základní práci s jazykem OpenSCAD. Zejména se zaměříme na vytváření geometrických primitiv (`objekt`), aplikací geometrických transformací na objekty (`transformace`) a kombinování objektů do složitějších objektů pomocí množinových operací (`operace`). Jazyk OpenSCAD ovšem nabízí i složitější konstrukce běžné v jiných programovacích jazycích, jako jsou proměnné, konstrukce pro řízení toku dat (cykly a větvení), možnost tvorby vlastních funkcí, knihoven a jiné.
Pro vytváření objektů používáme příkazy, které se skládají ze jména (to většinou odpovídá názvu objektu, který chceme vytvořit), jenž je následované parametry určující vlastnosti vytvářeného objektu, například specifikujeme jeho velikost. Parametry píšeme do kulatých závorek přímo za název příkazu.

```
objekt(parametry);
```

Na vytvořené objekty je možné aplikovat geometrické transformace. U transformací je možné v kulatých závorkách specifikovat tuto transformaci (například o kolik se má objekt posunout, otočit či zvětšit nebo zmenšit).[13] Na objekt je možné aplikovat více transformací, ty se píší za sebou a oddělují se mezerou.

[13] Transformaci je možné aplikovat na více objektů, v takovém případě se objekty za transformaci zapisují do složených závorek.

```
transformace(parametry_transformace) objekt(parametry);
```

Objekty kombinujeme pomocí operací následovně:

```
operace(){
  objekt_1(parametry_1);
  ...
  objekt_n(parametry_n);
}
```

Za názvem operace jsou ve složených závorkách vypsány všechny objekty, na které se má operace aplikovat.
Výsledkem aplikace transformací a operací na objekty vznikají nové

objekty, na které je možné aplikovat další transformace, případně je možné je použít v dalších operacích. Konkrétní názvy objektů a operací spolu s parametry, které je možné zadat, si ukážeme v následujících kapitolách.
Mimo výše zmíněné prvky je možné do kódu psát i *komentáře*. Vše, co je na řádku za `//` nebo mezi `/*` a `*/`, je překladačem ignorováno.

Shrnutí

V této kapitole jsme si stručně představili jazyk a vývojové prostředí OpenSCAD.

Kontrolní otázky
Odpovězte na následující otázky:

1. Co je to OpenSCAD?
2. Jak píšeme komentáře v jazyce OpenSCAD?

Cvičení

Úkol 4
Nainstalujte si vývojové prostředí OpenSCAD.

Úkol 5
Otevřete vestavěný příklad `logo.csad`, který najdete v menu `Soubor` → `Příklady` → `Základy`. Vyzkoušejte si práci s vývojovým prostředím.

Trojrozměrné objekty

V této kapitole se zaměříme na tvorbu jednoduchých trojrozměrných objektů. Ukážeme si vestavěná geometrická primitiva, ale i to, jakým způsobem můžeme do zdrojového kódu importovat objekty vytvořené v jiném modelovacím nástroji.

Geometrická primitiva

V této části si popíšeme základní geometrická primitiva, která je možné použít k vytváření modelů. Těmito primitivy budou krychle, koule, válec a mnohostěn.

Krychle

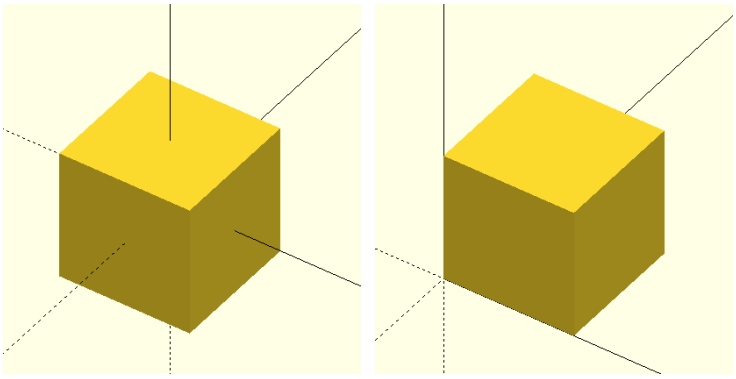
Krychli je možné vytvořit pomocí příkazu `cube()`. V kulatých závorkách lze specifikovat velikost krychle (parametr `size`) a také, kde se má vytvořit (parametr `center`). Parametry se oddělují čárkou a nezáleží na pořadí v jakém je uvádíme.

```
cube(size=1,center=false);
```

Parametr `center` může být nastaven na hodnotu `true` (pravda) nebo `false` (nepravda). Pokud je nastaven na `true`, pak je krychle vytvořena tak, že její střed leží v počátku souřadného systému. Pokud je nastavena na `false`, pak tam leží její vrchol. Na obrázku 7a je výsledek krychle vytvořené s parametrem `center=true` a na obrázku 7b `center=false`.
Parametr `size` určuje velikost vytvářené krychle. Velikost může být zadána jedním číslem (v tom případě bude vytvořena krychle o zadané délce stran), nebo trojicí čísel v hranatých závorkách `[x,y,z]`. Vytvořeným tělesem je pak kvádr s délkami stran x, y a z.
Ani jeden z parametrů není povinný a je možné jej vynechat. V takovém případě je použita *výchozí hodnota*. Výchozí hodnota parametru `size` je `1`, parametr `center` má výchozí hodnotu `false`.

Průvodce studiem
Krychle (pravidelný šestistěn nebo také hexaedr) je trojrozměrné těleso, jehož stěny tvoří 6 stejných čtverců.

Kvádr je trojrozměrné těleso — rovnoběžnostěn, jehož stěny tvoří šest pravoúhlých čtyřúhelníků. Má tři skupiny rovnoběžných hran shodné délky.



(a) S parametrem `center=true` (b) S parametrem `center=false`

Obrázek 7: Krychle.

Následující příkazy vytvoří stejné krychle.

```
cube(size=[1,1,1],center=false);
cube(size=1,center=false);
cube(center=false,size=1);
cube(size=1);
cube();
```

Jak je vidět na předchozím příkladu, parametry můžeme zadávat v libovolném pořadí. V případě, že jsou hodnoty parametrů zadávány ve správném pořadí (jak určuje dokumentace), můžeme vynechat jejich název a zapsat jen hodnoty. Viz následující příkaz.

```
cube(1,false);
```

Válec

Válec se vytváří příkazem `cylinder()`. Při vytváření válce je třeba specifikovat jeho výšku (parametr `h`) a poloměr (parametr `r`) nebo průměr (parametr `d`)[14] podstavy. Místo parametru `r` je možné použít parametry `r1` a `r2`. Ty určují poloměr každé podstavy válce zvlášť.[15] Obdobně je možné nahradit `d` parametry `d1` a `d2`. Na obrázku 8 můžeme vidět válec, komolý jehlan a jehlan vytvořené pomocí příkazu `cylinder()`.

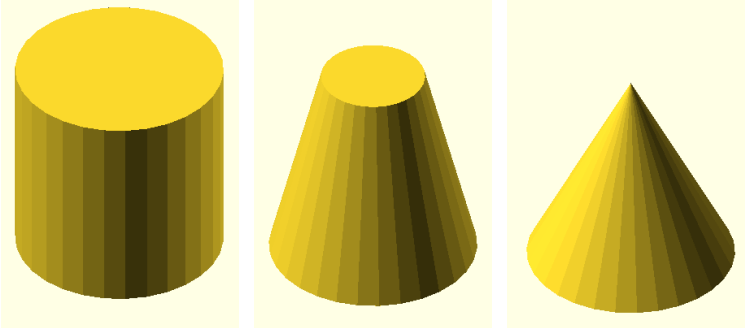
Průvodce studiem
Rotační válec je těleso, které je vymezeno dvěma rovnoběžnými kruhovými podstavami a pláštěm.

[14] Pokud jsou předány oba parametry, pak je `d` ignorován.

[15] Je zřejmé, že se v tom případě nejedná o válec, ale o komolý kužel.

Úkol 6
Jak vytvoříme pomocí příkazu `cylinder()` kužel?[16]

[16] Nastavíme poloměr (průměr) jedné z podstav na 0.



(a) Válec (b) Komolý kužel (c) Kužel

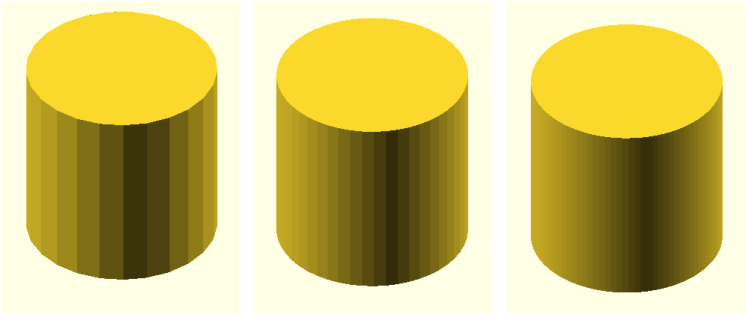
Obrázek 8: Válec.

Průvodce studiem
Rotační kužel je těleso vzniklé rotací pravoúhlého trojúhelníku kolem přímky *o*, na které leží jedna jeho odvěsna.

Pozici, kde se válec vytvoří se stejně jako u krychle specifikuje parametrem `center`. Další parametry, kterými je možné ovlivnit vzhled válce, jsou parametry `$fa`, `$fs` a `$fn`. Všechny tyto parametry ovlivňují přesnost kruhové podstavy. Čím je jejich hodnota vyšší, tím je kruh přesnější.[17] Specifikuje se vždy jen jeden z nich.[18] Na obrázku 9 můžeme vidět válce vytvořené s hodnotou parametru `$fn` nastavenou po řadě na 25, 50 a 100.

[17] Parametr `$fa` představuje minimální úhel každého fragmentu, parametr `$fs` minimální délku oblouku a parametr `$fn` počet fragmentů v 360°.

[18] Pokud není zadán žádný z parametrů `$fa`, `$fs`, `$fn`, je použita výchozí hodnota parametru `$fa`, kterou je hodnota 12, v důsledku toho výsledná podstava není kruhová.



(a) `$fn=20` (b) `$fn=50` (c) `$fn=100`

Obrázek 9: Válec vytvořený s různými hodnotami parametru `$fn`.

Pro vytvoření válce o výšce 1 a poloměru 1 použijeme následující příkaz.

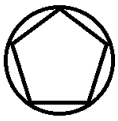
```
cylinder(h=1,r=1,$fa=12);
```

Pokud zvolíme nízkou hodnotu parametru `$fn`, například

```
cylinder(h=5,d=10,$fn=5);
```

vytvoří se pravidelný pětiboký hranol, který je vepsán do válce s průměrem 10. Podstavu tohoto hranolu můžete vidět na obrázku 10. Toto může vést k problémům, pokud je válec při vytváření složitějších objektů použit pro vytvoření díry. Vzniká tak totiž menší díra, než by bylo potřeba.

Obrázek 10: Podstava pětibokého hranolu vepsaného do válce.



Úkol 7
Vytvořte pravidelný čtyřstěn s délkou hrany 10 jednotek.
Řešení naleznete v kapitole *Řešení vybraných úkolů*.

Koule

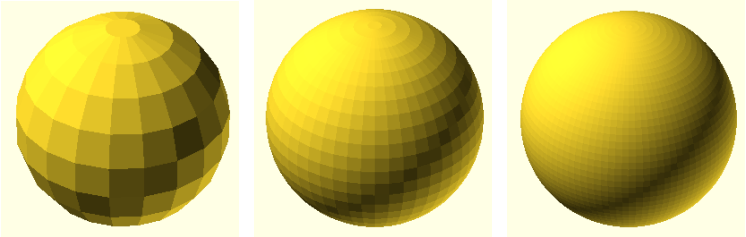
Koule se vytvoří příkazem `sphere()`. Velikost koule je možné specifikovat poloměrem (parametr `r`) nebo zadáním velikosti jejího průměru (parametr `d`).[19] Koule s poloměrem 1 se vytvoří příkazem

[19] Oba parametry nelze použít zároveň. Pokud jsou nastaveny oba, je parametr `d` ignorován.

```
sphere(r=1);
```

Kouli o poloměru 10 můžeme vidět na obrázku 11. Pokud není zadán ani poloměr ani průměr, pak se vytvoří koule s poloměrem velikosti 1. Koule nemá parametr `center` a vždy se vytvoří se středem v počátku. Stejně jako u válce můžeme přesnost koule ovlivnit parametry `$fa`, `$fs` a `$fn`.

Průvodce studiem
Koule je prostorové těleso tvořené množinou všech bodů (trojrozměrného euklidovského) prostoru, jejichž vzdálenost od zadaného bodu (středu) je nejvýše rovna zadanému poloměru.



(a) `$fn=20` (b) `$fn=50` (c) `$fn=100`

Obrázek 11: Koule vytvořená s různými hodnotami parametru `$fn`.

Mnohostěn

Průvodce studiem
Mnohostěn, také polyedr je trojrozměrné geometrické těleso, jehož povrch se skládá z konečně mnoha stěn tvořených mnohoúhelníky.

Nejobecnějším geometrickým primitivem v OpenSCAD je mnohostěn. Mnohostěn se vytváří příkazem `polyhedron()`. Je zadán mno-

žinou vrcholů (parametr `points`) a seznamem trojúhelníků (parametr `faces`),[20] které tvoří jeho povrch.
Každý vrchol je zadán trojicí `[x,y,z]` představující jeho souřadnice v trojrozměrném prostoru. Množina vrcholů je jejich výčet v hranatých závorkách.

[20] Od verze 2014.03 to nemusí být jen trojúhelníky, ale obecné mnohoúhelníky. Je nutné, aby vždy všechny vrcholy mnohoúhelníka ležely v jedné rovině. Proto je lepší používat jen trojúhelníky, kde je tato vlastnost splněna vždy.

```
points=[[x1,y1,y1],[x2,y2,z2],...,[xn,yn,zn]];
```

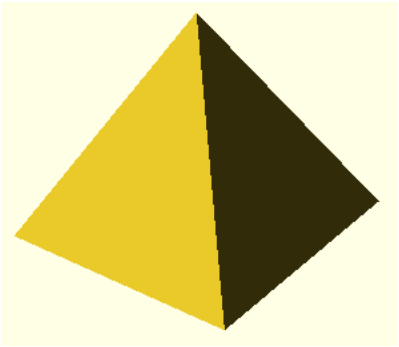
Každý trojúhelník je zadán trojicí vrcholů `[v1,v2,v3]`,[21] kde `v1`, `v2`, `v3` odkazují na vrcholy v množině `points`, kde jsou vrcholy očíslovány od `0` do `n-1` (`n` představuje počet vrcholů ve výčtu). Tomuto pořadí říkáme *index*. Trojice `[0,1,2]` představuje trojúhelník s vrcholy, jejichž souřadnice jsou uloženy ve výčtu vrcholů na indexech po řadě 0, 1 a 2, tedy první tři vrcholy.
Mnohostěn se vytvoří příkazem:

[21] V případě mnohoúhelníků se nejedná o trojice, ale obecně o n-tice.

```
polyhedron(points=[...],faces=[...]);
```

Následující příkaz vytvoří čtyřboký jehlan, který je zobrazen na obrázku 12.

Obrázek 12: Jehlan vytvořený pomocí mnohostěnu.



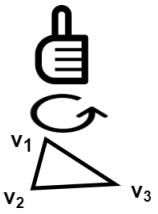
```
polyhedron(
   points=[[5,5,0],[5,-5,0],[-5,-5,0],[-5,5,0],[0,0,10]],
   faces=[[0,1,4],[1,2,4],[2,3,4],[3,0,4],[1,0,3],[2,1,3]]);
```

Úkol 8
Pomocí `polyhedron()` vytvořte pravidelný čtyřstěn s délkou hrany 10 jednotek.

Řešení naleznete v kapitole *Řešení vybraných úkolů*.

U objektů vytvářených pomocí `polyhedron()` je třeba si dávat pozor na pořadí vrcholů jednotlivých trojúhelníků (v parametru `faces`). Vrcholy trojúhelníku se při pohledu z venku na objekt zadávají proti směru hodinových ručiček. Můžeme také říct, že jsou uspořádány podle pravidla pravé ruky, které je znázorněno na obrázku 13 a říká: pokud pokrčené prsty pravé ruky ukazují směr zadaných vrcholů, pak palec ukazuje směrem ven.
Další na co je třeba si dát pozor je, že vytvářený objekt musí být možné zkonstruovat. Konkrétně, modely musí být uzavřené, bez děr a otevřených vnitřků.[22] Modely nesmí obsahovat dotýkající se stěny, hrany a vrcholy.[23]

Obrázek 13: Pravidlo pravé ruky.

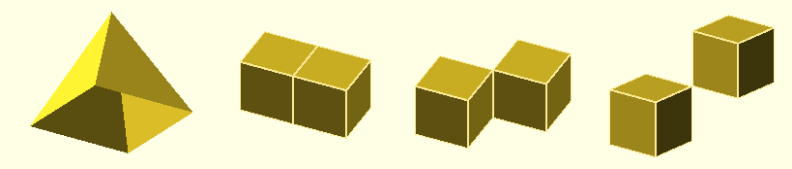


[22] Této vlastnosti se říká, že jsou tělesa *watertight*.

[23] Tělesa neobsahující dotýkající se stěny, hrany a vrcholy se nazývají *manifoldy*.

Na obrázku 14 jsou zobrazena tělesa, která nesplňují výše uvedené podmínky. První těleso obsahuje díru v podstavě,[24] další po řadě obsahují dotýkající se stěnu, hranu a vrchol.

[24] Tento výsledek obdržíme, pokud z předchozího příkladu odstraníme poslední dva trojúhelníky.
`polyhedron(points=[[5,5,0],[5,-5,0],[-5,-5,0],[-5,5,0],[0,0,10]], faces=[[0,1,4],[1,2,4],[2,3,4],[3,0,4]]);`



Obrázek 14: Chybně zadané mnohostěny.

Úkol 9
Pomocí `polyhedron()` vytvořte objekt, který nesplňuje některou z výše uvedených vlastností.

OpenSCAD editor nabízí možnost zkontrolovat správnost kódu. Najdeme jí v menu `Design/Zkontrolovat správnost`.

Importování objektů

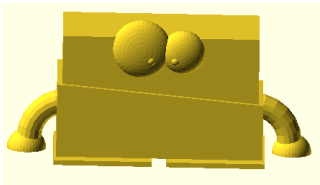
OpenSCAD umožňuje importování tvarů vytvořených v jiných programech, které jsou uložené ve formátu STL (s příponou `.stl`),[25] případně v případě novějších verzí OpenSCAD ve formátech AMF a 3MF. K importování objektů slouží příkaz `import()`. Nejdůležitějším parametrem je parametr `file`, kterým zadáváme cestu k již hotovému modelu.[26]
Na obrázku 15 můžeme vidět výsledek po importování souboru `robot.stl`[27] pomocí následujícího příkazu.

[25] STL je jeden z nejběžněji používaných formátů pro ukládání 3D objektů používaný zejména ve 3D tisku.

[26] Dalšími parametry jsou například parametry `convexity`, `layer` a další.

[27] Je potřeba zadat absolutní případně relativní adresu souboru.

```
import(file="robot.stl");
```



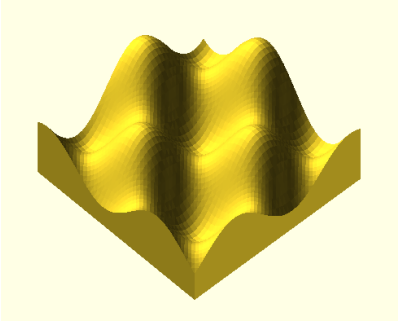
Obrázek 15: Robot importovaný do OpenSCAD.

Výšková mapa

Příkazem `surface()` je možné v OpenSCAD načíst informaci z textového souboru, nebo z `.png` obrázku a zobrazit tuto informaci jako výškovou mapu. Parametry, které tento příkaz přijímá, jsou `file`, kterým se specifikuje cesta k souboru, `center`, který má stejný význam jako u dříve zmíněných objektů, a `invert`, kterým se ovlivňuje to, jak jsou předaná data načtena. U textových dat však tento parametr nemá význam. Kromě zmíněných parametrů je zde i parametr `convexity`, kter má význam jen v náhledu, na finální renderování vliv nemá, nebudeme se jím tedy zabývat. Jak již bylo řečeno, výškovou mapu můžeme získat z textového souboru. Předpokládáme, že v něm je uložená matice hodnot, které jsou oddělené mezerou nebo tabulátorem. Každý prvek matice představuje konkrétní z-ovou hodnotu bodu daného indexem v matici. Na obrázku 16 můžeme vidět výsledek, který získáme při importování souboru, který obsahuje matici M, kde hodnota na indexu i, j je rovna $sin(i) * cos(j) * 10$, jde i i j jsou hodnoty od 1 do 10 s krokem 0.2.[28]

Obrázek (v `.png` formátu) je převeden na výškovou mapu tak, že se jako z-ová souřadnice bere intenzita (v případě šedotónového obrázku). Barevný obrázek je nejprve převeden do šedotónového a je zobrazena výsledná intenzita. Všechny hodnoty jsou škálovány na interval od 0 do 100. Parametrem `invert` ovlivníme to, jestli nejmenší hodnota bude odpovídat černé nebo bílé barvě. Na obrázku 17 můžeme vidět importovaný obrázek a výsledek.

Obrázek 16: Výšková mapa z textového souboru.



[28] V matlabu takovou matici získáme příkazem
```
surf = (sin(1:0.2:10)' *
cos(1:0.2:10)) * 10;
```

Obrázek 17: Výšková mapa obrázku.

Shrnutí

V této kapitole jsme si představily, jak vytvářet v OpenSCAD jednoduché trojrozměrné objekty, jako je koule, krychle válec a jiné. Také jsme si ukázali, jakým způsobem do kódu importovat již hotové trojrozměrné objekty.

Kontrolní otázky
Odpovězte na následující otázky:

1. Jakým příkazem vytvoříme kvádr?
2. Jak vytvoříme trojboký hranol?
3. Jakými parametry ovlivňujeme přesnost vykreslení některých objektů?

Cvičení

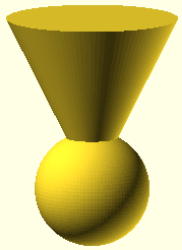
Úkol 10
Jak víme, příkaz `cylinder()` vytvoří n-boký (daný parametem `$fn`) hranol s podstavou vepsanou do kružnice zadané poloměrem (průměrem), kterým `cylinder()` specifikujeme. Jak vytvořit n-boký hranol, jehož podstava tuto kružnici opisuje?

Řešení naleznete v kapitole *Řešení vybraných úkolů*.

Úkol 11
Vymodelujte následující těleso. K jeho tvorbě není potřeba žádných transformací ani množinových operací.[29]

[29] OpenSCAD automaticky sjednocuje všechny objekty.

Úkol 12
Vymodelujte následující těleso. K jeho tvorbě není potřeba žádných transformací ani množinových operací.



Kombinace trojrozměrných objektů

V této kapitole si ukážeme, jakým způsobem můžeme již vytvořené objekty upravovat pomocí transformací a kombinovat je do složitějších objektů pomocí množinových operací.

Transformace

Na vytvořené objekty je možné aplikovat prostorové transformace. Mezi základní transformace patří změna velikosti, otočení a posunutí.

Průvodce studiem
Transformace (z lat. trans-formatio, pře-tvoření) se používá ve více významech. Obvykle je mylně používána jako podstatná změna i přes to, že pravý význam je postupná přeměna, přetváření.

Změna velikosti

Velikost objektu lze v jazyce OpenSCAD změnit pomocí příkazů `scale()` a `resize()`. Příkaz `scale()` se používá následovně.

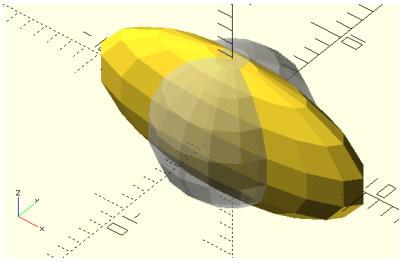
```
scale([x,y,z]) objekt();
```

`objekt()` v kódu představuje konkrétní objekt, například `cube()`. Předaný parametr `[x,y,z]` určuje v jakém měřítku se změní velikost objektu ve směru os x, y a z. Konkrétně, aplikací

```
scale([2,0.5,1])
```

na nějaký objekt se tento objekt ve směru osy x dvojnásobně zvětší, v ose y se jeho velikost zmenší na polovinu a v ose z zůstane velikost nezměněna. Viz obrázek 18, kde byla operace aplikována na kouli o poloměru 5 zobrazené světle šedou barvou.
Objektu můžeme měřítko změnit ve všech směrech stejně, v takovém případě můžeme místo trojice `[x,y,z]` použít jediné číslo.
Pokud chceme objektu změnit velikost na konkrétní hodnotu, použijeme příkaz `resize()` následovně.

Obrázek 18: Změna velikosti.



```
resize([x,y,z],auto=true) objekt();
```

Trojice `[x,y,z]` v tomto případě označuje konkrétní velikost objektu ve směru jednotlivých os. Pokud je nějaká z těchto hodnot rovna nule a je nastaven parametr `auto` na `true` bude objektu v tomto směru nastavena hodnota tak, aby byl zachován poměr zvětšení nebo zmenšení.[30] Pokud není `auto=true`, pak velikost tělesa ve směrech s nulovou hodnotou zůstává stejná.

[30] Pokud jsou zadaná dvě různá zvětšení, třetí se upraví podle většího z nich.

Otočení

Objekt je možné otočit pomocí příkazu `rotate()`. Otáčet je možné kolem všech os o stejný úhel, nebo okolo každé jinak. Použití je následující.

```
rotate(a,[x,y,z]) objekt();
rotate([x,y,z]) objekt();
```

V prvním případě parametr a představuje úhel ve stupních a druhý parametr `[x,y,z]` je trojice jedniček a nul, které představují okolo kterých os se má objekt otočit o úhel a (otáčí se okolo těch, které jsou rovny 1).
Druhé možné použití je takové, že se příkazu `rotate()` předává trojice úhlů, o kolik se má objekt otočit v každé z os. Příkaz

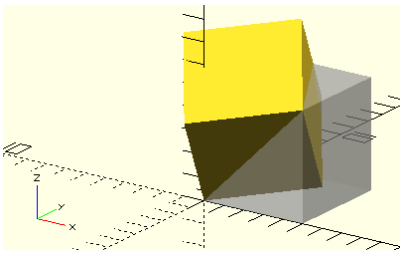
```
rotate(45,[1,0,1]) cube(size=1);
```

otočí krychli o 45 stupňů okolo os x a z (viz obrázek 19). Příkaz

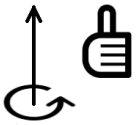
```
rotate([90,0,45]) cube(size=1);
```

otočí krychli kolem osy x o 90° a okolo osy z o 45°.
Otáčení okolo osy probíhá podle pravidla pravé ruky. Pokud palec ukazuje v kladném směru osy kolem které otáčíme, pak pokrčené prsty ukazují směr rotace, jak ukazuje obrázek 20.

Obrázek 19: Otočení.



Obrázek 20: Pravidlo pravé ruky.



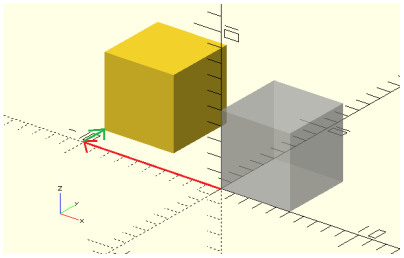
Posunutí

Pro posun objektu použijeme příkaz `translate()`, jehož použití je následující:

```
translate([x,y,z]) objekt();
```

Čísla `x`, `y` a `z` vyjadřují o kolik se má posunout objekt ve směru os x, y a z. Na obrázku 21 můžeme vidět krychli posunutou o 10 ve směru osy x, o 2 ve směru osy y a o 0 ve směru osy z.

Obrázek 21: Posunutí.



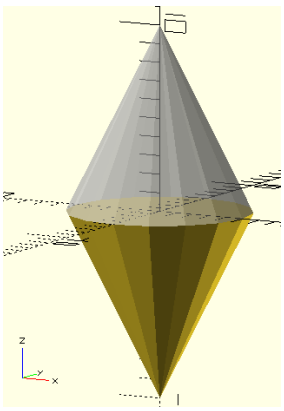
Zrcadlení

Kromě výše zmíněných transformací, můžeme objekt zrcadlit dle nějaké roviny pomocí transformace `mirror()`, jehož použití je následující:

```
mirror([x,y,z]) objekt();
```

Čísla `x`, `y` a `z` určují bod v prostoru, ze kterého je vedena přímka k počátku souřadného systému. Objekt je pak zrcadlen dle roviny kolmé k této přímce v počátku. Na obrázku 22 můžeme vidět jehlan zrcadlený dle XY roviny pomocí příkazu.

Obrázek 22: Zrcadlení.



```
mirror([0,0,1]) cylinder(r1=5, r2=0, h=10);
```

Afinní transformace

Nejobecnější způsob, jak transformovat objekty, je aplikovat na ně obecnou afinní transformaci. Tu zadáváme pomocí afinní matice, která obecně vypadá následovně.

$$\begin{bmatrix} \text{zvětšení X} & \text{zvětšení X zkosení Y} & \text{zvětšení X zkosení Z} & \text{posun X} \\ \text{zvětšení Y zkosení X} & \text{zvětšení Y} & \text{zvětšení Y zkosení Z} & \text{posun Y} \\ \text{zvětšení Z zkosení X} & \text{zvětšení Z zkosení Y} & \text{zvětšení Z} & \text{posun Z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tuto matici aplikujeme na objekt pomocí příkazu `multmatrix()` následujícím způsobem.

```
multmatrix(m=[...]) objekt();
```

Následující matice představuje afinní transformaci, která otočí objekt o 45° dle osy z a posune objekt o 10 ve směru osy x, o 20 ve směru osy y a o 30 ve směru osy z.[31]

$$\begin{bmatrix} cos(45) & -sin(45) & 0 & 10 \\ sin(45) & cos(45) & 0 & 20 \\ 0 & 0 & 1 & 30 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[31] To odpovídá následujícímu příkazu
`translate([10,20,30])`
`rotate([0,0,45]) objekt();`

Skládání transformací

Na objekt je možné aplikovat více transformací. Jednotlivé transformace se oddělují mezerou. V zápisu záleží na pořadí, ve kterém jsou operace zapsány. Operace se vykonávají v opačném pořadí, než jsou uvedeny, tedy zprava doleva. Například

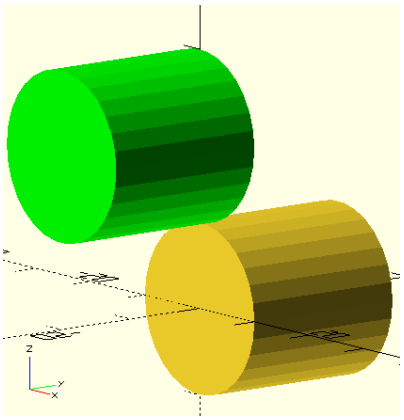
```
translate([0,20,0]) rotate([90,0,0]) cylinder(h=20,r=10);
```

nejprve otočí válec o 90° kolem osy x a pak jej posune ve směru osy y o 20 jednotek (na obrázku 23 žlutý válec).

```
rotate([90,0,0]) translate([0,20,0]) cylinder(h=20,r=10);
```

V tomto příkazu je nejprve válec posunut ve směru osy y a pak teprve otočen (na obrázku 23 zelený válec).

Obrázek 23: Skládání transformací.



Aplikace transformace na více objektů

Pro úplnost je potřeba dodat, že je možné transformace aplikovat na více objektů zároveň. V takovém případě je syntaxe následující.

```
transformace(...){
 objekt_1(...);
 objekt_2(...);
 ...
}
```

Například následující kód provede posun koule i krychle o 10 jednotek ve směru osy x.

```
translate([10,0,0]){
 sphere();
 cube();
}
```

Operace

V předchozích částech jsme se věnovali příkazům vytvářejícím jednoduchá geometrická primitiva a možnostem, jak změnit jejich polohu a velikost pomocí geometrických transformací. Nyní si ukážeme, jakým způsobem kombinovat více těles dohromady a vy-

Průvodce studiem
Množina je souhrn objektů, chápaný jako celek. Objekty množiny se nazývají prvky množiny.
Základními operacemi definovanými nad množinami jsou průnik, sjednocení, rozdíl, symetrický rozdíl a doplněk.

tvářet tak složitější tvary pomocí množinových operací sjednocení, průniku a rozdílu.
Pro sjednocení v OpenSCADu používáme příkaz `union()`, který se aplikuje na objekty, jež jsou uvedeny ve složených závorkách za tímto příkazem.[32] Pro sjednocení koule a krychle použijeme následující konstrukci.

[32] Vytvořené objekty se v OpenSCAD, pokud na ně není aplikovaná jiná množinová operace, automaticky sjednocují.

```
union(){
  cube(size=10,center=true)
  sphere(d=13);
}
```

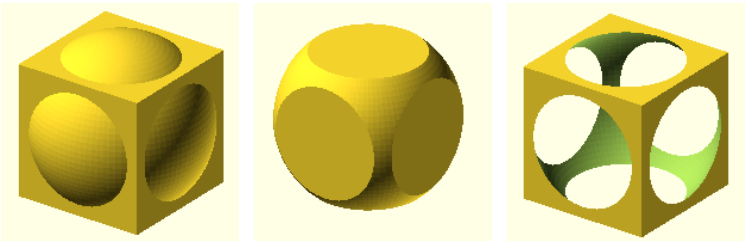
Pro průnik objektů se používá příkaz `intersection()`, obdobně, jako se používá `union()`.

```
intersection(){
  cube(size=10,center=true)
  sphere(d=13);
}
```

Pro rozdíl se používá příkaz `difference()`. Ten se aplikuje tak, že od prvního objektu, který je uveden ve výčtu ve složených závorkách, se odečítají všechny další objekty. Konkrétní příklad vypadá následovně.

```
difference(){
  cube(size=10,center=true)
  sphere(d=13);
}
```

Výsledky sjednocení, průniku a rozdílu krychle a koule z předchozích příkladů můžete vidět na obrázku 24.



(a) Sjednocení (b) Průnik (c) Rozdíl

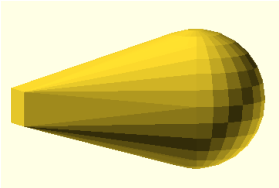
Obrázek 24: Množinové operace s krychlí a koulí.

Další způsoby kombinování objektů

Objekty dále můžeme kombinovat pomocí operátoru Minkovského nebo konvexním obalem. Na obrázku 25 můžeme vidět výsledek aplikace konvexního obalu na krychli a kouli, který získáme pomocí následujícího kódu.

```
hull(){
    translate([10,0,0]) sphere(8);
    translate([-10, 0, 0]) cube(4, center=true);
}
```

Obrázek 25: Konvexní obal.



Minkovského sumu válce a koule můžeme vidět na obrázku 26. Tento výsledek jsme získali pomocí následujícího kódu.

```
minkowski(){
    cylinder(r=15, h=20);
    sphere(4);
}
```

Obrázek 26: Minkovského suma

Ladění

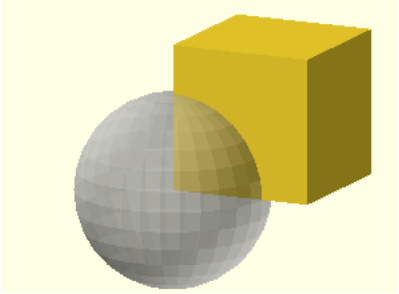
Při vytváření složitějších objektů se může stát, že člověk v modelu ztratí přehled. Zejména u odečítání objektů, případně při vytváření dutin, protože ty nejsou ve výsledku viditelné. V OpenSCAD existují modifikátory, pomocí kterých je možné ovlivnit to, co je v náhledu viditelné.

Modifikátor % (background)

Modifikátor % způsobí, že se daná část v náhledu zobrazí v průhledné barvě (při renderování se nezobrazí vůbec). Na obrázku 27 můžeme vidět výsledek následujícího kódu.

```
difference(){
  cube(15);
  %sphere(10);
}
```

Obrázek 27: Modifikátor %.

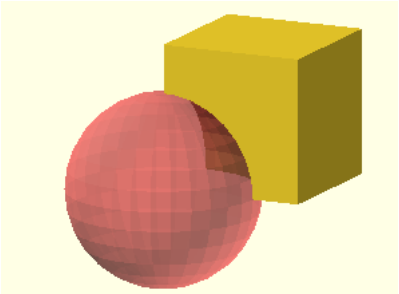


Modifikátor # (debug)

Modifikátor # způsobí, že se daná část v náhledu zobrazí v červené průhledné barvě (při renderování se vykreslí normálně). Na obrázku 28 můžeme vidět výsledek následujícího kódu.

```
difference(){
  cube(15);
  #sphere(10);
}
```

Obrázek 28: Modifikátor #.



Oproti předchozímu modifikátoru se kód zpracuje a je vidět, že je koule od krychle odečtena.

Modifikátor ! (root)

Modifikátorem ! učiníme označenou část kořenem CSG stromu (který představuje generované těleso) a ostatní části tělesa se ignorují (jak v náhledu, tak při renderování). Na obrázku 29 můžeme vidět výsledek následujícího kódu.

```
difference(){
  cube(15);
  ! union(){
        sphere(10);
        translate([5,5,5]) sphere(5);
  }
}
```

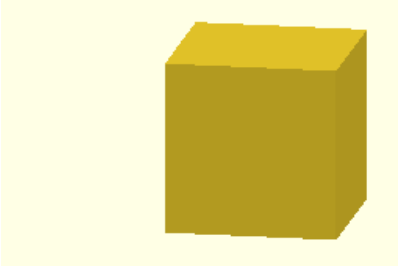
Obrázek 29: Modifikátor !.

Modifikátor * (disable)

Modifikátor * je pravým opakem předchozího modifikátoru. Díky němu označenou část CSG stromu odstraníme z výpočtu (opět jak v náhledu, tak při renderování). Na obrázku 30 můžeme vidět výsledek následujícího kódu.

```
difference(){
  cube(15);
  * union(){
        sphere(10);
        translate([5,5,5]) sphere(5);
  }
}
```

Obrázek 30: Modifikátor *.



Změna barvy

Chceme-li v náhledu[33] některou část objektu zobrazit jinou barvou, můžeme použít transformaci[34] `color()`. Barvu specifikujeme RGB hodnotou a průhledností. Možné zadání barev je následující.

[33] Informace o barvě je při renderování ignorována.

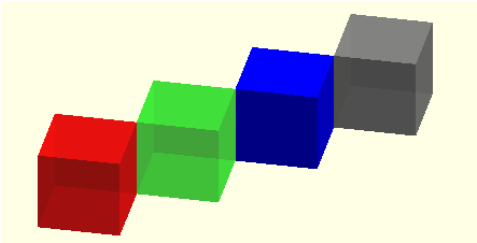
[34] Nejedná se o skutečnou transformaci, ale použití se s transformacemi shoduje.

```
color( c = [r, g, b, a] ) { ... }
color( c = [r, g, b], alpha = 1.0 ) { ... }
color( "#hexhodnota" ) { ... }
color( "nazev_barvy", 1.0 ) { ... }
```

Barvu tedy můžeme zadávat jako vektor čtyř hodnot v barevném modelu RGBA, v barevném modelu RGB a případně alfa složkou (defaultně je tento parametr nastaven na 1.0), hexadecimálním vyjádřením či názvem barvy.[35]
Na obrázku 31 můžeme vidět výsledek následujícího kódu.

[35] Seznam všech názvů barev můžeme nalézt například na `https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations#color`

```
color(c=[1,0,0], alpha=0.75) cube(5);
color(c=[0,1,0,0.5]) translate([5,5,0]) cube(5);
color("Blue") translate([10,10,0]) cube(5);
color("#888888e0") translate([15,15,0]) cube(5);
```



Obrázek 31: Změna barvy.

Shrnutí

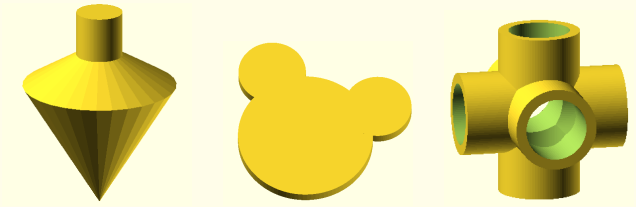
V této kapitole jsme si představili, jak vytvářet z jednoduchých objektů složitější tělesa za pomoci množinových operací a to konkrétně pomocí operací sjednocení, rozdílu a průniku. Také jsme si představili geometrické transformace jakými jsou například otočení, posunutí nebo změna měřítka. Navíc jsme si ve stručnosti ukázali možnosti, jak kód ladit ať už za pomoci modifikátorů, nebo změny barvy částí tělesa.

Kontrolní otázky
Odpovězte na následující otázky:

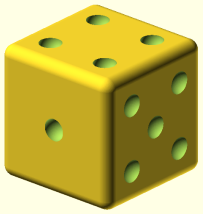
1. Co je to afinní transformace?
2. Jaké množinové operace podporuje OpenSCAD?
3. K čemu slouží modifikátory?
4. Jaký je rozdíl mezi transformacemi `resize()` a `scale`?

Cvičení

Úkol 13
Vytvořte tělesa podobná těm, které můžete vidět na obrázcích níže.

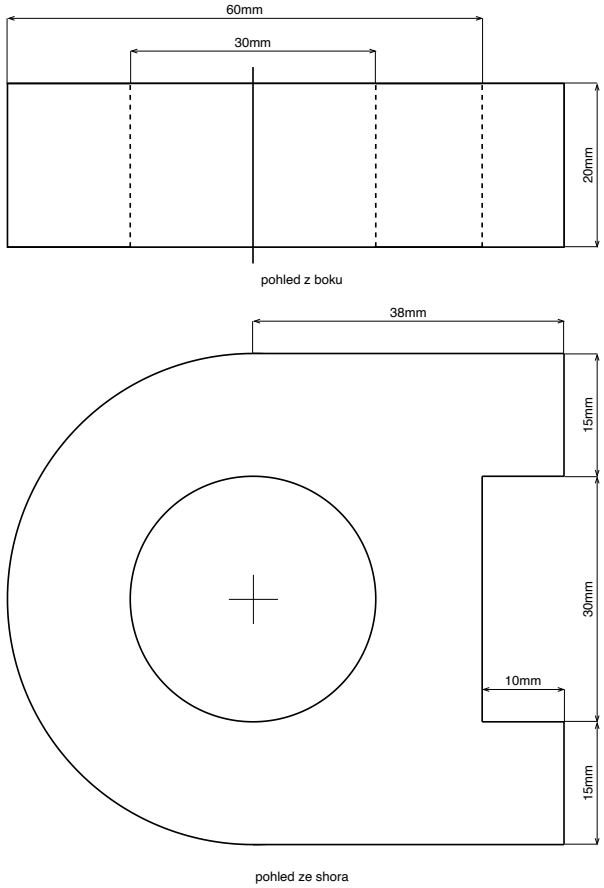


Úkol 14
Vymodelujte šestistěnnou hrací kostku podobnou té na obrázku níže.

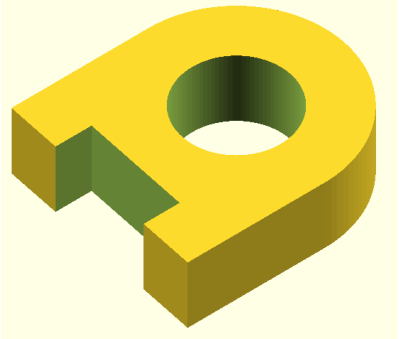


Řešení naleznete v kapitole *Řešení vybraných úkolů*.

Úkol 15
Vytvořte těleso podle níže uvedeného schématu. Výsledné těleso můžeme vidět na obrázku 32.



Obrázek 32: Výsledné těleso.



Dvourozměrné objekty

V této části se budeme věnovat dvourozměrným objektům a ukážeme si, jak z nich vytvořit objekty trojrozměrné.

Dvourozměrné objekty

Dvourozměrné objekty jsou objekty, které jsou nekonečně tenké, vytváří se v rovině os x a y (*XY rovina*) a v náhledu[36] se zobrazují s tloušťkou 1. Tyto objekty se vytváří pomocí klíčového slova, které označuje typ objektu, a v kulatých závorkách se specifikují vlastnosti těchto objektů stejně, jak tomu bylo v případě trojrozměrných objektů.

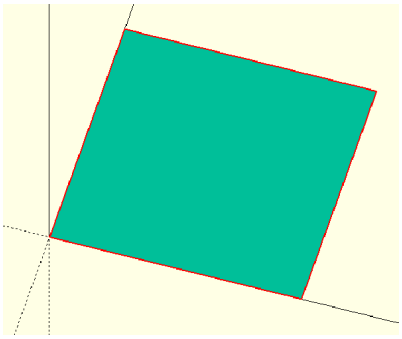
[36] po stisku F5.

Čtverec/obdélník

Mezi základní dvourozměrné objekty patří čtverec, který se vytváří příkazem `square()` (viz obr 33). Parametry tohoto příkazu jsou stejné jako u krychle, tedy `size` (velikost), která může být buď jedno číslo, nebo dvojice v hranatých závorkách,[37] a `center`, ovlivňující umístění čtverce.

```
square(size=1,center=false);
```

Obrázek 33: Čtverec.



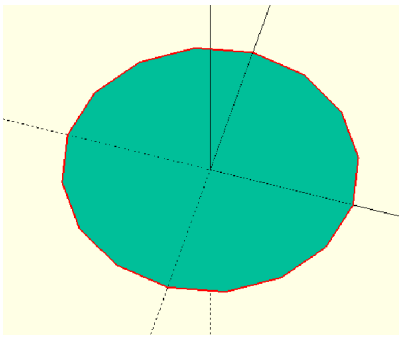
[37] V takovém případě se nejedná o čtverec, ale o obdélník.

Kruh

Dalším základním objektem je kruh, který vytvoříme příkazem `circle()` (viz obr 34). Parametry, které mu můžeme zadat, jsou obdobné jako u koule, tedy můžeme specifikovat jeho velikost parametry `r` (poloměr) nebo `d` (průměr) a přesnost ovlivnit parametry `$fa`, `$fs` nebo `$fn`.

```
circle(r=1);
```

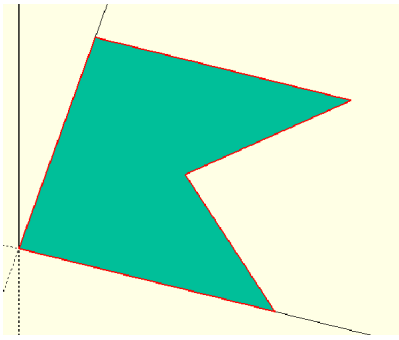
Obrázek 34: Kruh.



Mnohoúhelník

Nejobecnějším dvourozměrným objektem je mnohoúhelník. Příkaz vykreslující mnohoúhelník je `polygon()` (viz obr 35). Jako parametry se mu předávají vrcholy (`points`), které jsou zadávány dvojicí `[x,y]` představující x-ovou a y-ovou souřadnici bodu, a cesty (`paths`), což jsou sekvence indexů (indexy v hranatých závorkách) odkazujících do množiny bodů představující hranice mnohoúhelníku. Pokud není cesta zadána, pak je hranice dána pořadím vrcholů v parametru `points`. Cest obecně může být více (v parametru `path` je jejich výčet). První cesta představuje vnější hranici mnohoúhelníku, všechny další jsou chápány jako díry v něm. Mnohoúhelník musí být vždy uzavřený, což znamená, že poslední vrchol je roven prvnímu a je doplněn automaticky. Níže uvedený příkaz vykreslí čtverec se stranou délky 10.

Obrázek 35: Mnohoúhelník.

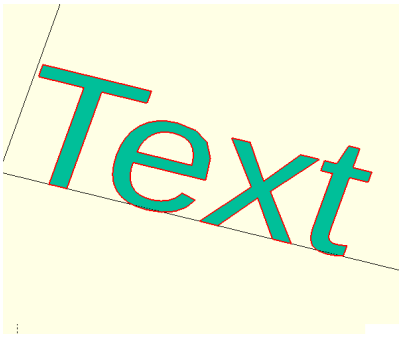


```
polygon(points=[[0,0],[0,10],[10,0],[10,10]], paths
    =[[0,1,3,2]]);
```

Text

Dalším dvourozměrným objektem je objekt text, který vytvoříme příkazem `text()` (viz obrázek 36). Parametr `text` udává text, který chceme zobrazit. Vzhled textu je možné měnit dalšími parametry. Například `size`, `font`,[38] `spacing`, `direction`.

Obrázek 36: Text.



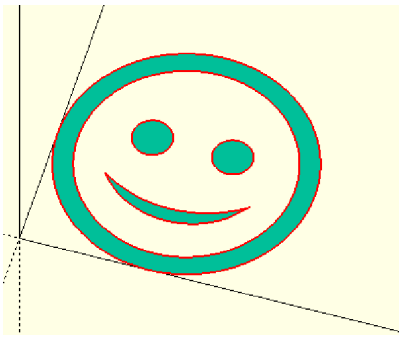

```
text("Text");
```

[38] Pokud si nejste jisti názvy fontů, které máte v počítači instalované, je možné je zobrazit `Nápověda` → `Seznam písem`.

Import 2D objektů

Stejně jako tomu bylo u 3D objektů, můžeme do OpenSCAD importovat 2D objekty vytvořené pomocí jiného softwaru ve formátu `.dxf` a v novější verzi také ve formátu `.svg`. Ty importujeme následujícím příkazem (výsledek viz obrázek 37).

Obrázek 37: Import.



```
import("smajlik.svg");
```

Transformace a množinové operace

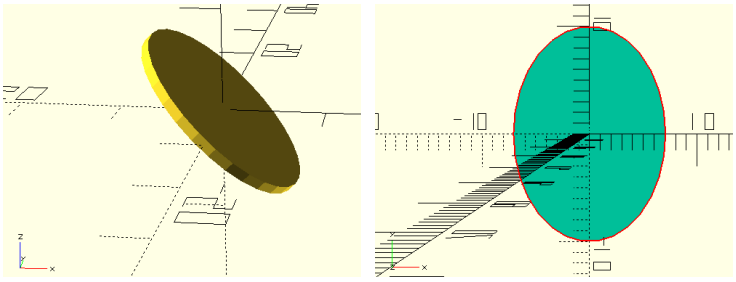
Na dvourozměrné objekty můžeme aplikovat transformace a množinové operace obdobně jako na trojrozměrné objekty.[39] Viz podkapitola *Transformace* předchozí kapitoly.
Pokud budeme chtít například posunout čtverec ve směru osy x o 10 a ve směru osy y o 5, použijeme příkaz `translate` následovně:

[39] Transformace jako jsou změna velikosti a posun je možné provádět jen ve směrech os x a y a rotace má smysl jen kolem osy z.

```
translate([10,5]) square(size=1);
```

Přesto, že nemá význam aplikovat na objekty transformace tak, aby výsledný objekt neležel v rovině XY, možné to je. V takovém případě se výsledný objekt v náhledu zobrazí mimo tuto rovinu (viz obrázek 38a), při renderování se však objekt do této roviny promítne (viz obrázek 38b). Obrázek 38 jsme získali pomocí následujícího kódu.

```
rotate([0,45,0]) circle(r = 10);
```

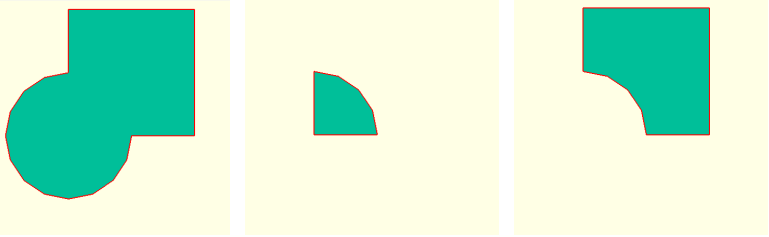


(a) Náhled (b) Renderování

Obrázek 38: Kruh, který neleží v rovině XY.

Jednotlivé objekty můžeme kombinovat pomocí množinových operací stejně jako trojrozměrné objekty tak, jak bylo uvedeno v podkapitole *Operace* předchozí kapitoly. Na obrázku 39 vidíme výsledky sjednocení, průniku a rozdílu čtverce a kruhu.

Úkol 16
Dvěma různými způsoby vytvořte elipsoid (2D objekt, jehož hranice je elipsa).



(a) Sjednocení (b) Průnik (c) Rozdíl

Obrázek 39: Množinové operace pro čtverec a kruh.

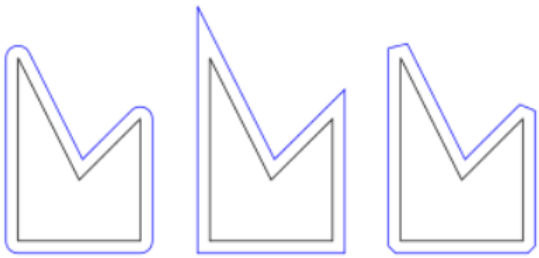
Offset

Navíc můžeme objekty zmenšovat a zvětšovat pomocí příkazu `offset()` s následovně.

```
offset(...) objekt();
```

Příkazu `offset()` jako parametr zadáváme `r` nebo `delta`, které určují, o kolik se má objekt zmenšit (záporné číslo) nebo zvětšit (kladné číslo). Rozdíly mezi těmito parametry je v tom, že při použití parametru `r` budou rohy výsledného objektu zakulacené, v případě parametru `delta` budou ostré. Dalším parametrem je `chamfer`. Tento parametr má význam pouze za použití parametru `delta` a určuje, zda mají být ostré rohy (`true`) ořezány, nebo ne (`false`).
Na obrázku 40 vidíme výsledky pro jednotlivé parametry s kladným číslem parametrů `r` a `delta`.[40]

[40] Černou barvou je naznačen původní objekt, na který byl aplikován offset, a modrou výsledný objekt.



(a) `r = x` (b) `delta = x` `chamfer = false` (c) `delta = x` `chamfer = true`

Obrázek 40: Offset.

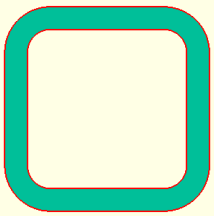
Na obrázku 41 vidíme výsledky pro záporné hodnoty parametrů `r` a `delta`.[41]

[41] Černou barvou je naznačen původní objekt, na který byl aplikován offset, a červenou výsledný objekt.

(a) `r = -x` (b) `delta = -x` `chamfer = false` (c) `delta = -x` `chamfer = true`

Obrázek 41: Offset.

Úkol 17
Vytvořte následující objekt.



Řešení naleznete v kapitole *Řešení vybraných úkolů*.

Vytvoření trojrozměrných objektů z dvourozměrných

Dvourozměrné objekty samy o sobě nemůžeme využít při konstrukci trojrozměrných modelů. Musíme z nich nejprve vytvořit trojrozměrné objekty. K tomu můžeme využít tzv. *extruzi*. Tento pojem by se dal přeložit do českého jazyka jako vytlačení. V OpenSCADu existují dva typy extruze a to lineární a rotační.

Průvodce studiem
Jako extruze (česky vytlačování) se obecně nazývá jakýkoli proces, ve kterém dochází k protlačování materiálu přes nějakou formu.

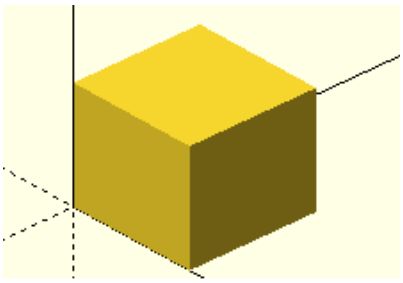
Lineární extruze

Lineární extruze se provádí příkazem `linear_extrude()`, který vyžaduje parametr `height`[42] a aplikuje se na dvourozměrný objekt následovně:

[42] Pokud není zadán, pak je nastaven na 1.

```
linear_extrude(height=1) objekt();
```

Obrázek 42: Lineární extruze.



Příkaz vytvoří trojrozměrný objekt, který bude mít podstavy rovny objektu `object()` a jeho výška bude dána předaným parametrem `height`. Spodní podstava bude ležet v rovině XY. Pokud bychom chtěli, aby rovina XY procházela středem vytlačeného objektu, můžeme nastavit parametr `center` na hodnotu `true` (defaultní hodnota je `false`). Následující příkaz vytvoří ze čtverce o straně 12 jednotek kvádr s výškou 10 jednotek. Výsledek je zobrazen na obrázku 42.

```
linear_extrude(height=10) square(size=12);
```

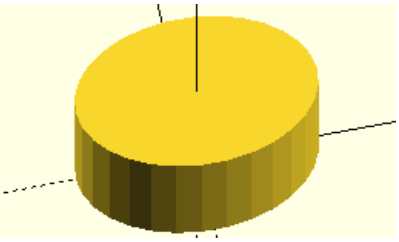
Pokud objekt, na který aplikujeme lineární extruzi, neleží v rovině XY, pak je objekt do této roviny promítnut[43] a až pak je na něj

[43] Jak už víme, do roviny XY je promítnut i samotný 2D objekt při renderování.

aplikována extruze. Na obrázku 43 vidíme výsledek získaný při překladu následujícího kódu.

```
linear_extrude(10) rotate([45,0,0]) circle(10);
```

Obrázek 43: Lineární extruze.



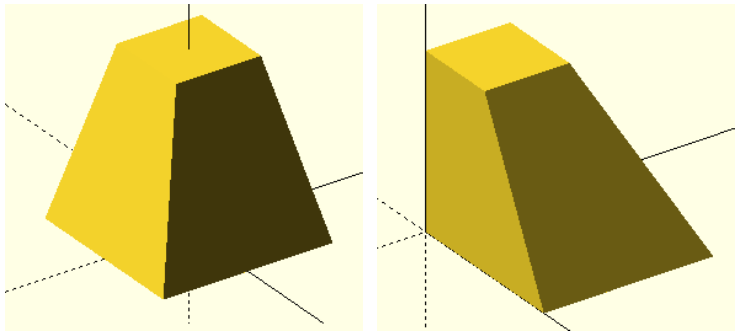
Extruze nemusí být pravidelná. Pokud použijeme v příkazu `linear_extrude()` parametr `scale`,[44] pak se změní měřítko horní podstavy v poměru zadaném tímto parametrem.

```
linear_extrude(height=1,scale=1) objekt();
```

Spodní podstava trojrozměrného objektu bude odpovídat objektu `objekt()`, horní podstava bude zvětšena `scale` krát.[45] V následujícím kódu nastavujeme parametr `scale` na jednu polovinu.

```
linear_extrude(height=10,scale=0.5) square(size=10);
```

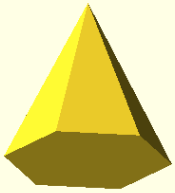
Extruze se sbíhá směrem k ose z (rozbíhá od osy z), viz obrázek 44. Na obrázku 44a je u čtverce, na který je lineární extruze aplikována, nastaven parametr center na `true`, na obrázku 44b ne.



(a) `square(10, center=true);` (b) `square(10);`

Obrázek 44: Lineární extruze.

Úkol 18
Pomocí lineární extruze vytvořte šestiboký jehlan.



Dalším parametrem ovlivňujícím extruzi je parametr `twist`. Ten představuje počet stupňů, o kolik se objekt během extruze otočí (otáčí se po směru hodinových ručiček, pokud chceme otáčet proti směru, použijeme zápornou hodnotu).[46]

```
linear_extrude(height=1,twist=0)
```

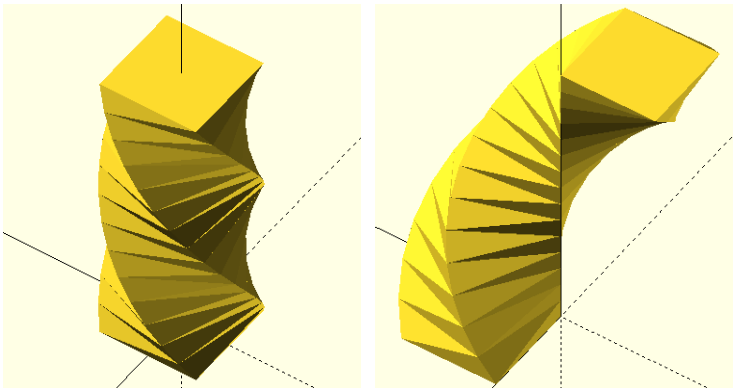
I v tomto případě probíhá otáčení kolem osy z. Na obrázku 45 můžeme vidět výsledek při nastavení parametru `twist` na 180. Line-

[44] Výchozí hodnota tohoto parametru je rovna 1. Proto pokud tento parametr nenastavíme, pak se velikost objektu během extruze nezmění.

[45] Během extruze se velikost objektu postupně mění.

[46] Výchozí hodnota tohoto parametru je rovna 0.

ární extruze je aplikována na čtverec se středem v počátku (obrázek 45a) i na čtverec se středem mimo počátek (obrázek 45b).

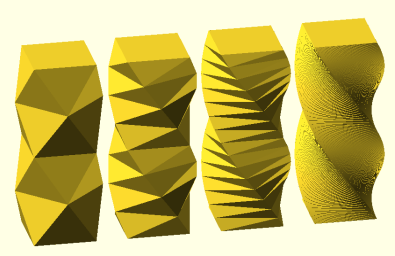


(a) `square(10, center=true);` (b) `square(10);`

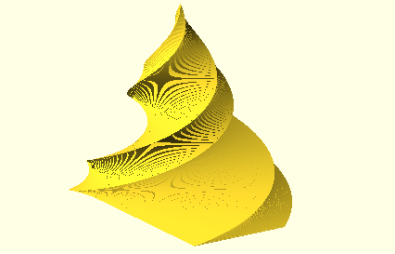
Obrázek 45: Lineární extruze.

Přesnost vykreslení můžeme ovlivnit parametry `$fa`, `$fn` a `$fs`. Na obrázku 46 můžeme vidět výsledky s postupně se zmenšujícím parametrem `$fa`. Parametry `twist` a `scale` můžeme kombinovat. Výsledek můžeme vidět například na obrázku 47.

Obrázek 46: Lineární extruze.



Obrázek 47: Lineární extruze.



Rotační extruze

Příkaz `rotate_extrude()` vytvoří trojrozměrný objekt tak, že otočí dvourozměrný objekt kolem osy z.[47]

```
rotate_extrude(angle=360,$fa=12) objekt();
```

Parametr `angle` určuje o kolik stupňů se má objekt otočit podle pravidla pravé ruky. Na obrázku 48 můžeme vidět výsledky pro následující kód, přičemž parametr `angle` postupně nabývá hodnot 90, 180 a 360.

```
rotate_extrude(angle=uhel) translate([2,0,0]) circle(r=1);
```

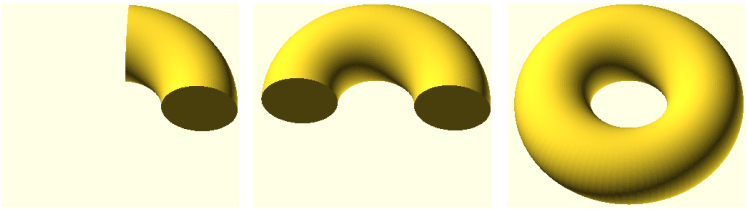
Pokud není zadán úhel, otočí se o 360°. Stejně jako tomu bylo u lineární extruze, pokud není objekt v rovině XY, nejprve se do této roviny promítne. Parametr `angle` se objevil v OpenSCAD až ve verzi 2019, ve starších verzích je vždy prováděna rotace o celých 360 stupňů.
Dvourozměrný objekt musí být celý vpravo[48] od osy y.[49] Pokud objekt protíná y-ovou osu, pak se objekt nevykreslí. Pokud se y-ové osy dotýká, musí se dotýkat linií, ne jen jedním bodem.[50]

[47] Objekty leží v rovině XY. Těleso vzniklo otočením objektu kolem osy y a pak bylo otočeno tak, jako by vzniklo otočením kolem osy z.

[48] Může být i vlevo, ale doporučuje se vpravo.

[49] Přesněji řečeno buď musí být x-ové souřadnice všech bodů objektu menší nebo rovny 0 nebo větší nebo rovny 0.

[50] Vznikl by objekt, který není manifold.



(a) `uhel = 90` (b) `uhel = 180` (c) `uhel = 360`

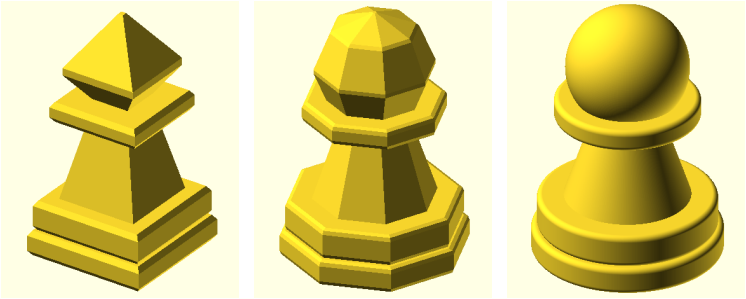
Obrázek 48: Rotační extruze.

Úkol 19
Jakým způsobem bychom vytvořili pomocí rotační extruze kouli?

Řešení naleznete v kapitole *Řešení vybraných úkolů*.

Parametry `$fa`, `$fn` a `$fs` můžeme ovlivnit přesnost vykreslení objektu.[51] Na obrázku 49 můžeme vidět výsledky pro různé hodnoty parametru `$fn`.

[51] Jak budou vypadat jednotlivé fragmenty během rotace.



(a) `$fn = 4` (b) `$fn = 8` (c) `$fn = 360`

Obrázek 49: Rotační extruze.

Vytvoření dvourozměrného objektu z trojrozměrného

Na závěr kapitoly si pro úplnost uvedeme možnost, jak z trojrozměrného objektu můžeme vytvořit objekt dvourozměrný. K tomu v OpenSCAD slouží příkaz `projection()`. Jedná se o projekci objektu do roviny XY. Tyto dvourozměrné objekty je pak možné exportovat do formátu `.dxf` nebo `.svg`.[52] Použití je následující.

[52] Do těchto formátů lze exportovat i dvourozměrné objekty uvedené dříve.

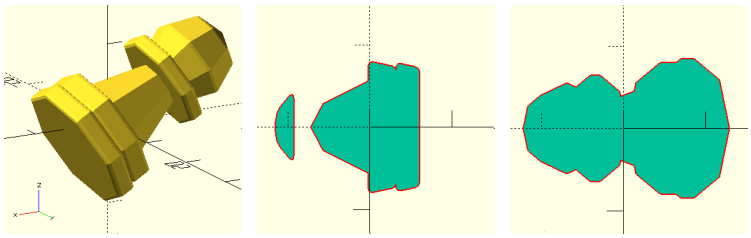
```
projection(cut=false) objekt();
```

Pokud je parametr `cut` roven `true`, budou výsledkem pouze body se $z = 0$,[53] pokud je roven `false` (což je defaultní hodnota) do roviny XY se promítnou všechny body i ty, co jsou nad nebo pod rovinou XY.[54] Na obrázku 50 můžeme vidět model, na který byla

[53] Jedná se o řez rovinou.

[54] Opravdová projekce tělesa.

aplikována projekce s různým parametrem `cut`.



(a) Originál (b) `cut = true` (c) `cut=false`

Obrázek 50: Projekce.

Shrnutí

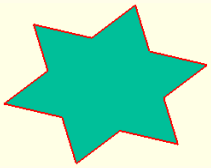
V této kapitole jsme si představili jiný způsob, jakým můžeme objekty vytvářet. Konkrétně jsme si ukázali, jak vytvářet dvourozměrné objekty, které na trojrozměrné převádíme pomocí extruze.

Kontrolní otázky
Odpovězte na následující otázky:

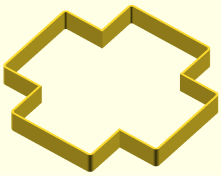
1. Co je to extruze?
2. Jaký je rozdíl mezi lineární a rotační extruzí?
3. K čemu slouží transformace offset?

Cvičení

Úkol 20
Vymodelujte šesticípou hvězdu.



Úkol 21
Vytvořte vykrajovátko ve tvaru kříže, podobné tomu na obrázku níže.

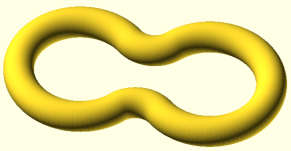


Úkol 22
Vymodelujte šestistěnnou hrací kostku podobnou té na obrázku níže.

Řešení naleznete v kapitole *Řešení vybraných úkolů*.

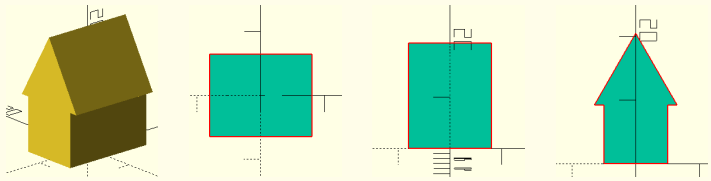
Úkol 23
Zkuste vymodelovat figurku pěšce podobou té na obrázku 49. Zkuste vymodelovat ve stejném duchu i ostatní figury.

Úkol 24
Zkuste vymodelovat následující objekt.



Řešení naleznete v kapitole *Řešení vybraných úkolů*.

Úkol 25
Pro libovolný objekt vytvořte pomocí projekce jeho půdorys[55], bokorys[56] a nárys.[57] Na obrázku níže vidíme postupně objekt, půdorys, bokorys a nárys.



[55] Pohled seshora.

[56] Pohled zboku.

[57] Pohled zepředu.

Úkol 26
Vytvořte válec na na něj umístěte text „ABC“. Viz následující obrázek.

Programování

Doposud jsme si ukázali základy, jak modelovat pomocí OpenSCAD. Zatím jsme ale nevyužili jeho plný potenciál. Jak už bylo řečeno dříve, OpenSCAD umožňuje používání cyklů pro opakování částí kódů a také můžeme kód větvit. Kromě toho si také vysvětlíme, jak se v OpenSCAD používají proměnné.

Hodnoty

OpenSCAD je dynamicky typovaný jazyk s pevnou množinou datových typů, ty nejsou pojmenované. Není možné zde vytvářet typy vlastní.
Hodnoty v OpenSCAD jsou buď čísla (například `42`), logické hodnoty (`true/false`), řetězce (`''text''`), rozsah[58] (například `[0 : 1 : 10]`), vektor hodnot (například `[1, 2, 3]`) nebo nedefinovaná hodnota (`undef`).

[58] Range.

Čísla

Čísla můžeme zapisovat v decimální[59] notaci obdobně jako v jiných jazycích. Například tedy

[59] Nepodporuje hexadecimální ani oktalovou notaci.

```
42
-1
0.5
1.234e+5
```

Kromě čísel v desítkové soustavě podporuje i některá pojmenovaná čísla, například `PI`.
Všechna čísla jsou reprezentovaná jako 64bitové číslo s plovoucí řádovou čárkou.[60] To má za následek to, že některá čísla není možné vyjádřit přesně (například `0.2`), ale některá ano (například `0.25`).[61]
Čísla, která jsou menší nebo větší, než je možný rozsah,[62] jsou brány jako mínus nebo plus nekonečno a vypisují se jako `-inf` nebo `inf`. `inf` však není konstanta a v případě potřeby je nutné si vytvořit

Průvodce studiem
Pohyblivou řádovou čárkou nebo plovoucí řádovou čárkou se rozumí způsob reprezentace čísel, která by byla moc malá nebo velká pro vyjádření v pevné řádové čárce. Čísla jsou obecně uložena jako určité množství platných číslic vynásobený exponentem. Základem exponentu bývá většinou 2 (což odpovídá dvojkové soustavě). Čísla, která mohou být v pohyblivé řádové čárce vyjádřena přesně, jsou ve tvaru $cislice \cdot zaklad^{exponent}$.

[60] Nerozlišujeme celá a desetinná čísla.

[61] 0.25 je možné vyjádřit jako $1/2^2$.

[62] Nejmenší reprezentovatelné číslo je přibližně $-1e308$ a největší $1e308$

proměnnou s hodnotou tak velkou, že se nedá reprezentovat. Například $1e200 * 1e200$.
Záporná čísla hodně blízké nule se reprezentují jako `-0`. Pro některé numerické operace je hodnota `-0` jiná než hodnota `0`, ale při porovnání jsou si tyto hodnoty rovny.
Na čísla můžeme aplikovat aritmetické operace `+` (sčítání), `-` (odčítání), `*` (násobení) a `/` (dělení). Dále lze použít goniometrické operace `cos()`, `sin()`, `tan()`,[63] a jiné matematické operace, jako jsou `abs()` (absolutní hodnota), `ceil()` (zaokrouhlení nahoru), `round()` (zaokrouhlení k nejbližší celé hodnotě), `floor()` (zaokrouhlení dolů), `sqrt()` (odmocnina) a další.[64]
Při provádění matematických operací můžeme dostat jako výsledek hodnotu not a number (v OpenSCAD se vypisuje jako `nan`). Ani `nan` není konstanta. Tuto hodnotu můžeme dostat například `0/0`. `nan` je jediná hodnota, která není rovna žádné jiné hodnotě kromě sebe sama.[65]

[63] `acos()`, `asin()`, `atan()`

[64] Seznam všech matematických operací najdeme v [2].

[65] Pro test, zda proměnná není hodnoty `nan` nemůžeme tedy použít `x == 0/0`, ale můžeme test provést takto `x!=x` (`x` nerovná se `x`).

Logické hodnoty

Logické hodnoty `true` a `false` se nejčastěji používají v podmínkách.[66]
Logické hodnoty, stejně jako je tomu v jiných programovacích jazycích, můžeme kombinovat pomocí logických operací `&&` (and), `||` (or) a `!` (not).
Jako hodnota nepravda se dále považuje prázdný řetězec (`'' ''`), prázdný vektor (`[]`) a kladná i záporná nula (`0` a `-0`). Zbylé hodnoty se považují za pravdu.[67]

[66] Později se podmínkám budeme věnovat více.

[67] A to i například `[false]`, `[ [ ] ]` nebo `0/0` (`nan`).

Řetězce

Řetězec je sekvence (unicode) znaků. Řetězce se zejména používají k pomocným výpisům,[68] specifikaci názvů souborů, které importujeme, nebo při tvorbě 2D objektu `text`.
Řetězce zapisujeme do uvozovek. Pokud potřebujeme použít v řetězci znak uvozovek, použijeme zpětné lomítko jako escape znak. Pro zpětné lomítko musíme jeho použití zdvojit. Mezi další speciální znaky patří `\n` pro nový řádek, `\t` pro tabulátor, `\r` pro carriage return, `\uxxxx` nebo `\Uxxxxxx`, kde `x` reprezentují čtyř nebo šestimístný unicode kód.
Například při překladu následujícího kódu dostaneme v konzoli výpis na obrázku 51.

[68] Jak už bylo řečeno, slouží k tomu příkaz `echo()`.

```
echo("\U01f436 jí \U01f9b4.");
```

Obrázek 51: Výpis v konzoli.

ECHO: "🐶 jí 🦴."

Pro převod jiné hodnoty na řetězec v OpenSCAD existuje funkce `str()`. Pokud je funkci předáno více argumentů, převede všechny

argumenty na řetězec a spojí je do jednoho řetězce.

Úkol 27
Vyzkoušejte následující kód a porovnejte jednotlivé výstupy.

```
cislo = 4;

echo("Toto je", cislo, 2);
echo(str("Toto je", cislo, 2));
```

Pro zjištění délky řetězce můžeme použít funkci `len()`. Pro převod Unicode čísla na znak a naopak můžeme použít funkce `chr()` respektive `ord()`.[69]

[69] Seznam všech funkcí pro práci s řetězci najdeme v [3].

Vektory

Vektor je kolekce hodnot, ty mohou být různých typů (čísla, vektory, řetězce, proměnné a logické hodnoty). Navíc to mohou být i výrazy, které se vyhodnotí jako nějaká hodnota.
Vektory jsou reprezentovány jako hodnoty uzavřené v hranatých závorkách (`[` a `]`) oddělené čárkami.
Například následující vektor[70]

[70] V kódu jsme vytvořili proměnnou `v`, které jsme přiřadili vektor. Proměnným se budeme více věnovat později.

```
v = [3, 5, [6,7], [[8,9],[10,[11,12],13]], "string"];
```

obsahuje pět hodnot.[71] Velikost vektoru můžeme získat pomocí funkce `len()`.[72]
Následující kód vypíše do konzole `5`.

[71] Dvě čísla, dva vektory a řetězec.

[72] Jak už víme, funkci `len()` můžeme použít i pro zjištění délky řetězce.

```
echo(len(v));
```

K jednotlivým prvkům vektoru se dostaneme pomocí indexů. Prvky indexujeme od 0 do $n-1$, kde n je počet prvků vektoru. Indexy zapisujeme do hranatých závorek.[73]
Prvek `[6,7]` vektoru `v` dostaneme následujícím způsobem.

[73] Stejným způsobem můžeme indexovat i jednotlivé znaky v řetězci.
`"text"[2]`
Vrátí prvek `x`.

```
v[2]
```

Tento vektor můžeme dále indexovat a dostat prvek `7`.

```
v[2][1]
```

Při překročení indexu mimo jeho meze dostaneme hodnotu `undef`.

Úkol 28
Pro následující vektor

```
v = [ [1], [], [2,3,4], "text", "x", [[5,6],[7,8,9],[[10,11],[12]]] ];
```

určete následující prvky a jejich délku.

```
v[0]
v[1]
v[5]
v[5][1]
v[5][2]
v[5][2][0]
v[5][2][0][1]
```

Jelikož se vektory v OpenSCAD používají hlavně pro specifikaci souřadnic bodů, nebo jako specifikace velikosti v jednotlivých osách, je možné k prvním třem prvkům přistupovat pomocí tečkové notace následovně.

```
v.x // to same co v[0]
v.y // to same co v[1]
v.z // to same co v[2]
```

Jednotlivé vektory můžeme použít jako prvky jiných vektorů. Pro spojení více vektorů[74] do jednoho můžeme použít funkci `concat()`[75] následovně.

[74] Libovolného počtu.
[75] Stejnou funkci můžeme použít pro spojení více řetězců.

```
concat([1, 2], [3, 4]); // [1 2 3 4]
```

Rozsah

Rozsah hodnot se používá nejčastěji v cyklech. Můžeme je vytvářet dvěma způsoby.

```
[start:konec]
[start:krok:konec]
```

I přes to, že je zapsán v hranatých závorkách, nejedná se o vektor. Pokud je `krok` kladný, měla by hodnota `start` být menší než `konec`.[76] Měli bychom se vyvarovat hodnotám `krok`, které není možné vyjádřit přesně v binární notaci desetinných čísel s pohyblivou řádovou čárkou. Pokud není `krok` zadán, bere se, že je roven 1.

[76] V případě, že použijeme notaci `[start:konec]` a bude `start` větší než `konec`, vypíše se warning a vytvoří se rozsah `[konec:start]`. V případě, že použijeme `[start:krok:konec]` a `start` bude větší než `konec`, vytvoří se prázdný rozsah.

Úkol 29
Podívejte se, jakým způsobem vypisuje příkaz `echo()` rozsahy.

Pro převod rozsahu na vektor hodnot můžeme použít funkci `each()`. Následující kód

```
[-2, each [1:2:5], each [6:-2:0], -1]
```

vytvoří vektor `[-2, 1, 3, 5, 6, 4, 2, 0, -1]`.

Funkce pro ověřování datových typů

V OpenSCADu existuje několik funkcí pro testování datových typů, které nám umožňují ověřit, jakého typu jsou dané hodnoty či proměnné. Tyto funkce jsou užitečné zejména při kontrole vstupních dat. Mezi nejpoužívanější funkce tohoto typu patří například `is_undef()`, kde testujeme, zda je hodnota definovaná, `is_num()` pro testování, zda je zadaná hodnota číslo, `is_list()` pro test, zda se jedná o vektor, `is_string()` a další.[77]

[77] Seznam všech funkcí můžeme najít v [4].

Proměnné

Hodnoty, se kterými v kódu pracujeme, mohou být uchovávány v *proměnných*. Každá proměnná má své jméno (identifikátor)[78] skrze něž můžeme s proměnnou pracovat. Hodnotu proměnné lze přiřadit následovně:

[78] Identifikátor může obsahovat pouze malé a velké znaky anglické abecedy, čísla a podtržítko. Existují i speciální proměnné, jejichž identifikátory začínají znakem `$`. Už jsme se s podobnými proměnnými setkali při definici hladkosti vykreslení (`$fn`, `$fs`, `$fa`). Více se jim budeme věnovat později.

```
promenna = hodnota;
```

V tomto výrazu `promenna` představuje název proměnné a `hodnota` nějakou konkrétní hodnotu. Například příkazem

```
a = 3;
```

vytvoříme proměnnou pojmenovanou `a`, které je přiřazena hodnota 3. Proměnným je možné přiřazovat hodnotu opakovaně. Při vyhodnocování programu má ale proměnná vždy poslední hodnotu, která jí byla přiřazena, a to v celém programu.[79] To znamená, že proměnné je hodnota nastavena při překladu, ne za běhu programu.

[79] OpenSCAD je funkcionální programovací jazyk.

Úkol 30
Zkuste následující kód přeložit v OpenSCAD. Než to ale uděláte, zamyslete se nad tím, jaký bude výsledek.

```
a = 3;
echo(a);
a = 4;
echo(a);
```

Výstupem je dvakrát vypsaná hodnota 4, která byla proměnné `a` přiřazena jako poslední.
Proměnné je možné přiřazovat hodnoty přímo, nebo hodnoty vypočítané, ale i hodnoty jiných proměnných.

```
a = 3; // proměnná a má hodnotu 3
b = 2 + 2; // proměnná b má hodnotu 4
c = b + 1; // proměnná c má hodnotu 5
```

Proměnná, které nebyla přiřazena hodnota, má hodnotu `undef`.

```
echo("Promenna a ma hodnotu ", a);
```

Předchozí kód vypíše „Promenna a ma hodnotu undef“.

Úkol 31
Vysvětlete proč následující konstrukce není v OpenSCAD v pořádku: `x = x + 1;`

Platnost proměnných

Jak již bylo řečeno, hodnota proměnné se nastavuje u při překladu a je rovna poslední přiřazené hodnotě. Toto však není úplně přesné. Platnost proměnné je dána lokálním rozsahem (scope). Jak víme, pokud chceme aplikovat operace nebo transformace na více objektů, tak jsme objekty uzavírali do složených závorek. A právě složené závorky vytváří nový scope uvnitř současného.[80] Proměnná má poslední hodnotu přiřazenou v současném scope, pokud zde proměnné hodnota přiřazená není, pak má hodnotu vnějšího.

```
a = 3;
echo(a);
translate([0, 0, 0]){
    a = 4;
    echo(a);
}
echo(a);
```

[80] Pozor! Pouze složené závorky nevedou k vytvoření nového scope. V následujícím kódu bude dvakrát vypsána hodnota 4.

```
a = 3;
echo(a);
{
    a = 4;
    echo(a);
}
```

V předchozím kódu se vypíše postupně hodnota 3, 4 a 3, protože proměnné a je ve vnějším scope nastavena hodnota 3, ale díky `translate()` vznikl nový scope, kde jí je přiřazena hodnota 4.
Případně je možné nový scope vytvořit pomocí `let()`,[81] které nastavuje svému podstromu hodnoty proměnných, které jsou mu předány. Používá se následujícím způsobem.

[81] Ve starších verzích OpenSCAD mu odpovídala funkce `assign()`.

```
a = 3;
echo(a);
let(a = 4){
    echo(a);
}
echo(a);
```

Opět budou postupně vypsány hodnoty 3, 4 a 3.

Cykly

Cykly se ve většině programovacích jazyků používají k opakovanému vykonání stejné části kódu. V OpenSCAD cykly fungují trochu jinak, díky tomu, že se nejedná o imperativní programovací jazyk. V cyklech zde „iterujeme“ přes nějaký rozsah nebo vektor hodnot a pro každou hodnotu provedeme nějaké akce.[82]
Základní syntaxe je následující:[83]

[82] Nejčastěji se vytvoří nějaký objekt nebo objekty.

[83] Pokud tělo cyklu obsahuje jen jediný příkaz, je možné vynechat složené závorky.

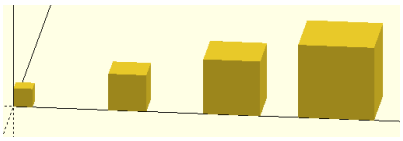
```
for(promenna = hodnoty){
    // telo cyklu
}
```

Pro každou hodnotu (z rozsahu nebo vektoru `hodnoty`) se vytvoří nový scope, kde je tato hodnota navázaná na proměnnou `promenna`, a je vykonáno tělo cyklu. V těle cyklu nemá tedy význam provádět nějaké výpočty, které by se měly použít dále za tělem cyklu, protože, jak už víme, v nadřazeném scope hodnoty nevidíme.
V následujícím kódu se vytvoří čtyři krychle s postupně se zvětšující velikostí a posunem ve směru osy x. Výsledek můžeme vidět na obrázku 52.

Úkol 32
Kolikrát proběhne cyklus, pokud bude `promenna = [1,n]` pro nějaké `n`?

```
for (i=[0:3])
   translate([i*5,0,0])
     cube(i+1);
```

Obrázek 52: For cyklus.

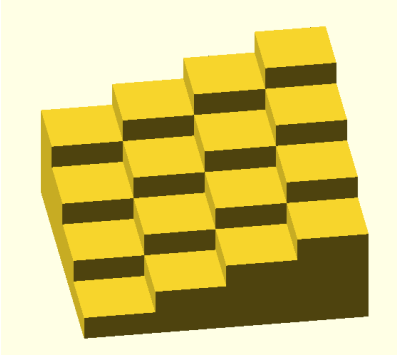


For cykly je možné do sebe vnořovat.[84] Příkladem může být následující kód, jehož výsledek můžeme vidět na obrázku 53.

[84] Je možné v těle cyklu použít další for cyklus.

```
for (i=[0:3]){
    for (j=[0 : 3]){
        translate([3*i,3*j,0]) cube([3,3,i+j+1]);
    }
}
```

Obrázek 53: For cyklus.



Protože je v prvním for cyklu pro každou hodnotu z rozsahu `[0 : 3]` vytvořen nový scope, kde vykonáváme tělo cyklu, ve kterém opět máme for cyklus, je možné předchozí kód zapsat zkráceně následovně.

```
for (i=[0:3], j=[0 : 3]){
    translate([3*i,3*j,0]) cube([3,3,i+j+1]);
}
```

Úkol 33
Porovnejte chování následujícího kódu v OpenSCAD a jak by se podobný kód choval například v jazyce C.

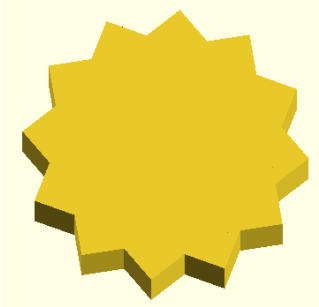
```
for (i=[0:3], j=[0 : 3]){  // for(i=0, j=0; i<=3 && j<=3; i++, j++){
    echo(i+j);             //       printf("%i\n", i+j);
}                          // }
```

Jak už víme, v OpenSCAD se objekty, pokud není řečeno jinak, sjednocují. Výsledkem vykonávání for cyklu je tedy sjednocení objektů, které vznikly v jednotlivých iteracích.
Pokud bychom chtěli vytvořit průnik těchto objektů, musíme použít místo for cyklu `intesection_for()`. Ten má stejnou syntaxi jako `for()`.

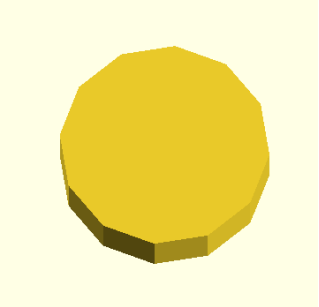
```
intersection_for(promenna = hodnoty){
    // telo cyklu
}
```

Na obrázku 54 můžeme vidět výsledky stejného kódu[85] za použití `for()` a `intersection_for()` cyklů.

[85] V tělě cyklu vytváříme krychle, které otáčíme kolem osy z.



(a) `for()`



(b) `intersection_for()`

Obrázek 54: Porovnání cyklů.

Podmínky

Obdobně jako v jiných programovacích jazycích, je možné v OpenSCAD na základě podmínek rozhodovat, jakou část kódu vykonat a jakou ne. Na rozdíl od nich, ale je potřeba mít stále na paměti, jak se zde pracuje s proměnnými. Syntaxe je následující.[86]

[86] Pokud blok vykonávající se při (ne)splnění podmínky obsahoval pouze jeden příkaz (jiný než přiřazení hodnoty nějaké proměnné), není nutné závorky používat.

```
if (podminka){
    // kod vykonavajici se pri splneni prodminky
}

if (podminka){
    // kod vykonavajici se pri splneni prodminky
}
else{
    // kod vykonavajici se pri nesplneni prodminky
}
```

`podminka` je většinou nějaký logický výraz.[87] Například můžeme použít operátor `==` (rovná se), `!=` (nerovná se), `<`, `<=` a jiné. Jak je vidět výše, kód vykonávající se při (ne)splnění podmínky je vlastně nový scope, takže pokud bychom uvnitř měnili hodnoty proměnných, vně nebude mít tato změna vliv, viz následující příklad. Vypíší se postupně hodnoty −5, 5 a −5.

[87] Dříve jsme si uvedli, že jako nepravda se navíc počítá i 0, prázdný vektor nebo prázdný řetězec.

```
b = -5;
echo(b);
if (b < 0){
    b = 5;
    echo(b);
}
echo(b);
```

Blok kódu vykonávaný při (ne)pravdivosti podmínky může obsahovat další větvení. Syntaxi můžeme tedy rozšířit následovně.

```
if (podminka_1){
    // kod vykonavajici se pri splneni prodminky 1
}
else if(podminka_2){
    // kod vykonavajici se pri nesplneni prodminky 1 ale pri splneni podminky 2
}
else{
    // kod pri nesplneni ani jedne podminky
}
```

Kromě klasického větvení pomocí `if()` je možné používat také ternární operátor podmínky, který většina zná z jiných jazyků.

```
podminka ? hodnota_pravda : hodnota_nepravda
```

Pokud je splněna `podminka`, vrátí se hodnota `hodnota_pravda` jinak je vrácena `hodnota_nepravda`. Následuje konkrétní příklad, který do proměnné `abs_a` uloží absolutní hodnotu proměnné `a`.

```
abs_a = a < 0 ? -a : a;
```

Shrnutí

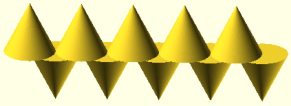
V této kapitole jsme si ukázali, jak v OpenSCAD pracovat s proměnnými, jak pomocí větvení ovlivnit jaká část kódu bude vykonávána a také, jak vypadají cykly v jazyce OpenSCAD.

Kontrolní otázky
Odpovězte na následující otázky:

1. Jaké datové typy se používají v OpenSCAD?
2. Jak indexujeme vektory?
3. Jaké cykly můžeme v OpenSCAD používat a jak se liší?

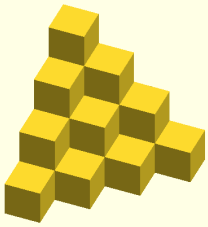
Cvičení

Úkol 34
Pomocí jednoho cyklu vymodelujte následující objekt.



Úkol 35
Pomocí cyklu vymodelujte následující objekty.

Úkol 36
Vymodelujte následující objekt, kde počet řad bude záviset na proměnné `n`.

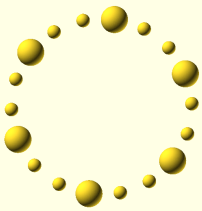


Řešení naleznete v kapitole *Řešení vybraných úkolů*.

Úkol 37
Vymodelujte následující objekt představující RGB krychli. Krychle se skládá z menších krychlí s délkou strany 10.[88]

[88] Barva se projeví pouze v náhledu. Viz kapitola Kombinace trojrozměrných objektů část Ladění.

Úkol 38
Pomocí jednoho cyklu vymodelujte následující objekty.



Funkce a moduly

V této kapitole si ukážeme, jak vytvářet uživatelsky definované funkce, které se v OpenSCAD chovají trochu jinak, než funkce v jiných programovacích jazycích, a vlastní moduly. Také si jen ve stručnosti ukážeme, jak vytvářet vlastní knihovny a jak používat již vytvořené.

Funkce

Funkce v jazyce OpenSCAD nejsou klasické funkce, jak je známe z jiných programovacích jazyků. Jsou to operace, které pracují s předanými hodnotami a vypočítají z nich hodnotu novou, kterou vrací. Syntaxe je následující.

```
function nazev( parametry ) = vysledna_hodnota ;
```

`nazev` je identifikátor funkce, pro který platí stejné omezení, jako pro identifikátory proměnných. `parametry` jsou názvy parametrů, které jsou odděleny čárkou.[89] Parametrům funkce je možné přiřadit defaultní hodnotu tak, že za jeho název přidáme operátor přiřazení a hodnotu. `vysledna_hodnota` je výraz, který vypočítá výslednou hodnotu.[90]
Platnost funkce je v rámci scope, ve kterém je definovaná. Pozor na možnost přepsání vestavěných funkcí.
Příklad použití následuje.

```
function funkce1() = [1, 2, 3];
function funkce2(x, y) = x + y;
function funkce3(x = 1) = 2 * x;

echo(funkce1());  // [1, 2, 3]
echo(funkce2(2, 3));  // 5
echo(funkce3());  // 2
echo(funkce3(2));  // 4
```

[89] Funkce nemusí mít žádné parametry, v takovém případě je závorka prázdná.

[90] Tou může být i vektor.

Pokud je ve funkci nastavena defaultní hodnota parametru, pak je možné funkci zavolat bez argumentů a je použita tato hodnota. Viz příklad funkce `funkce3()`.
Hodnoty, se kterými funkci voláme jsou postupně navázány na parametry, pomocí kterých je funkce volána. Na následujícím kódu si ukážeme další chování argumentů funkcí.

```
function funkce(x, y = 5) = x - y;

echo(funkce());  // chyba, protoze x je undef
echo(funkce(2));  // -3 x = 2, y = 5
echo(funkce(2, 2));  // 0
```

Pokud zavoláme funkci s žádnou hodnotou, vyhodnocení skončí chybou,[91] protože proměnná `x` není ve funkci nastavena. Při volání funkce s jednou hodnotou, je tato hodnota navázána na proměnnou `x` a proměnné `y` se přiřadí hodnota defaultní. V případě volání se dvěma argumenty se na `x` naváže první z nich a na `y` druhý.
Pozor ale na případ, kdy je nastavená defaultní hodnota prvnímu parametru místo druhého. Viz následující příklad.

[91] Ve skutečnosti je to jen varování a výsledkem je hodnota `undef`.

```
function funkce(x = 5, y) = x - y;

echo(funkce());  // chyba, protoze y je undef
echo(funkce(2));  // chyba, protoze y je undef, x = 2
echo(funkce(y = 2));  // 3 x = 5, y = 2
echo(funkce(y = 2, x = 3));  // 1 x = 3, y = 2
```

V takovém případě není možné volat funkci s jednou hodnotou, protože ta se naváže na proměnnou `x` (přepíše se její defaultní hodnota). V kódu ale demonstrujeme další chování a to to, že můžeme předané hodnoty pojmenovat a předávat tak hodnoty parametrům v jiném pořadí.[92]
V definici funkcí můžeme používat i proměnné, které funkci nepředáváme viz následující kód.

[92] Stejné chování jsme viděli při vytváření objektů.

```
function funkce1(x) = x - y;
function funkce2(x) = x - z;

z = 3;
echo(funkce1(5));  // chyba, protoze y je undef
echo(funkce2(5));  // 2 x = 5, z = 3
echo(funkce2(5, z = 5));  // 0 x = 5, z = 5
```

Při volání funkce `funkce1()` dojde k chybě, protože proměnná `y` je `undef`. Při prvním volání `funkce2()` je vrácena hodnota 2, protože

proměnná z je ve scope, kde je definovaná funkce rovna hodnotě 3. Při druhém volání funkce `funkce2()` dostaneme výsledek 0, protože jsme funkci předali i hodnotu z.[93]

[93] Toto volání vypíše warning, že z není specifikováno jako parametr funkce.

Rekurzivní funkce

OpenSCAD podporuje rekurzivní volání funkce. Pro koncovou podmínku se využívá ternárního operátoru podmínky. Následuje konkrétní příklad pro výpočet faktoriálu.

```
function faktorial(n) = (n == 0 ? 1 : n * faktorial(n-1) );
```

OpenSCAD má svůj vnitřní rekurzivní limit. V případě mnohonásobného volání skončí chybou. Vhodnější je používat koncovou rekurzi (tail-recursion), díky které můžeme výrazně zvýšit možný počet volání.

```
function faktorial(n, vysledek = 1) = (n == 0 ? vysledek : faktorial(n-1, vysledek) * n );
```

Anonymní funkce

Od verze *2021.01* OpenSCAD umožňuje vytvářet také anonymní funkce. Syntaxe je následující.

Průvodce studiem
Anonymní funkce (*lambda funkce*) jsou nepojmenované funkce, které umožňují práci s funkcemi jako s objekty. Tedy je možné je předávat do funkcí, nebo je z funkcí vracet.

```
function ( parametry ) vysledna_hodnota ;
```

Následuje konkrétní příklad použití.

```
f = function (x) 2 * x ;

echo(f(5)); // 10
```

V OpenSCAD je možné vytvářet funkce vyšších řádů, tedy funkce, které vrací funkce, nebo funkce berou jako argument. Viz následující příklady.

```
function op(typ) = (typ == "soucet") ? function (x,y) x + y : function (x,y) x - y;

echo(op("soucet")(5,3)); // vypise 8
echo(op("rozdil")(5,3)); // vypise 2
```

```
soucet = function (x,y) x + y;
rozdil = function (x,y) x - y;

function fun(f, x, y) = f(x , y);

echo(fun(soucet,5,3)); // vypise 8
echo(fun(rozdil,5,3)); // vypise 2
```

Moduly

Moduly v jazyce OpenSCAD jsou ekvivalenty maker nebo funkcí v jiných programovacích jazycích. Používání modulů nám umožní vytvářet znovupoužitelné kódy. Pomocí modulů můžeme vytvářet buď objekty, nebo operace. Syntaxe pro vytváření modulů je následující.

```
module nazev (parametry){
    // telo modulu
}
```

`nazev` představuje identifikátor modulu, pro který platí stejné podmínky, jako pro identifikátory proměnných. Pozor na předefinování již existujících modulů. `parametry` představují parametry, se kterými se budou moduly volat. Ty jsou od sebe oddělené čárkami a je možné jim přiřadit defaultní hodnotu, obdobně, jako v definicích funkcí. V těle modulů může být vše, co vně, je potřeba stále pamatovat na rozsah platnosti proměnných. Z modulů neexistuje možnost, jak vrátit nějakou hodnotu.

Objektové moduly

Pomocí modulů můžeme vytvářet objekty, které definujeme pomocí geometrických primitiv či jiných dříve definovaných modulů. Konkrétní příklad můžeme vidět v následujícím kódu.

```
module box (a=5){
    cube(size = a, center=true);
}
```

Samotná definice modulu nic nevytvoří. Pro vytvořené objektu je nutné modul zavolat pomocí jeho jména a případně parametrů. S objekty pak můžeme dále pracovat úplně stejně, jako s již předdefinovanými geometrickými primitivy, tedy můžeme na ně aplikovat transformace a kombinovat je pomocí operací. Pro volání modulů s parametry platí stejné vlastnosti, jako při volání funkcí. Pokud má

parametr defaultní hodnotu, nemusíme mu předávat hodnotu při volání, pokud dodržíme pořadí parametrů, nemusíme uvádět jejich názvy, při uvádění názvů nemusíme dodržovat jejich pořadí a pod. Viz následující příklady volání modulu box.

```
box();
box(a = 10);
translate([1,1,1]) box(1);
```

Moduly představující operace

V těle definující modul můžeme pomocí `children()` přistupovat k objektu (objektům)[94] uvedeným za voláním tohoto modulu. Uvnitř modulu na tento objekt můžeme aplikovat libovolné transformace či operace. Viz následující konkrétní příklad.

[94] Říkáme mu *potomek*, odtud název příkazu `children()`.

```
module posun_z(z = 0){
    translate([0, 0, z]) children();
}

posun_z(10) cube(1);
```

Modul `posun_z` definuje transformaci posunutí v ose z. Tato transformace se aplikuje na libovolný objekt, na který je modul aplikovaný. V našem případě na krychli.
Pokud je za modulem ve složených závorkách objektů více,[95] je tato operace (modul) aplikován na všechny. Viz následující kód.

[95] Jejich počet je uložen ve speciální proměnné `$children`.

```
posun_z(10){
    cube(10);
    sphere(5);
}
```

K jednotlivým potomkům je možné přistupovat pomocí indexů (od 0 do `$children` −1). Indexovat můžeme jedním číslem, nebo rozsahem indexů, případně jejich vektorem.

```
children();          // vsichni potomci
children(index);     // jeden konretni potomek
children(rozsah);    // vsichni potomci v rozsahu indexu
children(vektor);    // potomci s indexy ve vektoru
```

Úkol 39
Vytvořte modul, který po aplikaci na objekty tyto objekty postupně posune vždy o 10 ve směru osy x.[96]

[96] První posune o 10, druhý o 20 a tak dál.

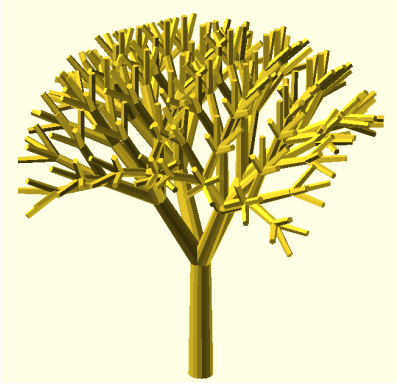
Rekurzivní moduly

Stejně jako funkce, můžeme vytvářet rekurzivní moduly. I zde je potřeba pamatovat na koncovou podmínku, aby nedocházelo k nekonečné rekurzi. V následujícím kódu můžeme vidět konkrétní příklad rekurzivního modulu. Výsledný objekt je vidět na obrázku 55.

```
 module strom(velikost, n){
    if (n > 0){
       cylinder(r1=velikost/10, r2=velikost/12, h=velikost);
       translate([0,0,velikost])
            for(iter = [0 : 4]){
                rotate([30,0,iter*90])
                    strom(0.75*velikost, n-1);
            }
    }
}

strom(50,  5);
```

Obrázek 55: Rekurzivní modul.



Funkce `parent_module()`

Proměnná `$parent_modules` obsahuje počet modulů v zásobníku instancí. Tento zásobník reprezentuje hierarchii instancí modulů, a je důležité poznamenat, že nezohledňuje, kde byly moduly definovány, ale pouze kde byly instanciovány.
`parent_module(i)` vrací název modulu, který je `i` úrovní nad současným modulem. Viz následující příklad.

```
module jedna(){
    children();
}

module dva(){
    children();
}

module tri(){
    children();
}
```

```
jedna() dva() tri() echo($parent_modules);  // vypise 3
jedna() dva() tri() echo(parent_module(0));  // vypise tri
jedna() dva() tri() echo(parent_module(1));  // vypise dva
jedna() dva() tri() echo(parent_module(2));  // vypise jedna
```

Tato funkce je užitečná při konstrukci Bill Of Materials (BOM), protože umožňuje snadno sledovat, z jakých modulů se skládá daný modul, a tím pádem můžete efektivně řídit hierarchii a závislosti mezi moduly.

Průvodce studiem
Bill of Materials (BOM), česky „kusovník“, je seznam všech komponentů, materiálů a dílů potřebných k výrobě určitého produktu nebo zařízení. Tento dokument slouží jako důležitý nástroj ve výrobních procesech a může zahrnovat názvy komponentů, jejich množství či popis. U složitějších produktů může být BOM strukturovaný do různých úrovní, aby ukazoval vztahy mezi hlavními díly a jejich podkomponenty.

Knihovny

Pro vložení externích souborů existují v OpenSCAD dva příkazy.

```
include <nazev_souboru>
use <nazev_souboru>
```

`nazev_souboru` je cesta k souboru,[97] který chceme do kódu vložit. Soubory se hledají nejprve ve složce, kde je uložen současný zdrojový kód, ve složce knihoven v instalaci OpenSCAD a pak ve složce, která je uložena v systémové proměnné `OPENSCADPATH`. Zatímco příkaz `include` se chová jako by se zkopíroval celý obsah souboru do zdrojového kódu, `use` vkládá pouze definice modulů a funkcí.[98] Vkládání kódu pomocí `include` nám umožňuje specifikovat defaultní hodnoty proměnných přímo v knihovně (kódu, který vkládáme). Tyto hodnoty můžeme v hlavním kódu přepsat. Jak už víme, proměnná má jen jednu hodnotu, po celou dobu běhu programu, při vícenásobném přiřazení se bere poslední přiřazená hodnota a tato hodnota je proměnné nastavena už při jejím prvním vytvoření.[99] Viz následující kód `lib.scad`.

[97] Cesta může být zadána relativně, nebo absolutně.

[98] Nevykonají se žádné jiné příkazy.

[99] Ne při prvním použití.

```
i=1;
k=3;
module x(){
    echo("i=",i,"j=",j,"k=",k);
}
```

Při vložení do kódu do souboru

```
j=4;
include <lib.scad>
x();
i=5;
x();
```

```
k=j;
x();
```

dostaneme následující výstupy.

```
ECHO: "i=", 5, "j=", 4, "k=", 4
ECHO: "i=", 5, "j=", 4, "k=", 4
ECHO: "i=", 5, "j=", 4, "k=", 4
```

Použijeme-li pro vložení kódu `use`, vloží se pouze definice funkcí a modulů. Nevyhodnotí se žádné jiné příkazy, tedy ani případné `include` nebo `use`.

Shrnutí

V této kapitole jsme si ukázali, jak v OpenSCAD vytvářet vlastní funkce a moduly a díky tomu vytvářet znovupoužitelný kód. Také jsme si ukázali, jak do kódu vkládat jiné již vytvořené moduly a funkce.

Kontrolní otázky
Odpovězte na následující otázky:

1. Jaký je rozdíl mezi funkcí a modulem?
2. Je `cube()` také modul?
3. Jak do kódu vkládat jiné soubory?

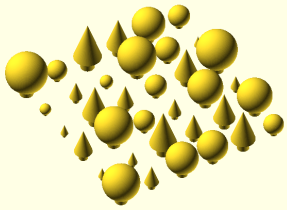
Cvičení

Úkol 40
Vytvořte funkci `delka()`, která vrátí délku předaného řetězce. Ve funkci není možné použít funkci `len()`.[100]

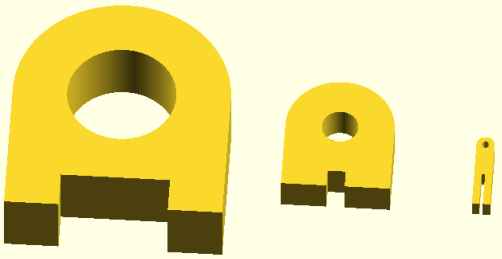
Řešení naleznete v kapitole *Řešení vybraných úkolů*.

[100] Vzpomeňte si, co dostaneme za hodnotu, když budeme indexovat řetězec mimo jeho délku.

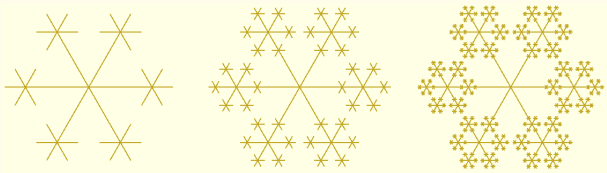
Úkol 41
Vytvořte dva moduly – `listnaty_strom` a `jehlicnaty_strom` a pomocí cyklu a generátoru náhodných čísel vytvořte náhodný les podobný tomu na následujícím obrázku.



Úkol 42
Vytvořte modul, pro generování tělesa z úkolu 15. Některé rozměry použijte jako parametry modulu, aby bylo možné modelovat tělesa podobná. Viz následující obrázek.



Úkol 43
Vytvořte modul, pro generování pravidelných vloček, viz následující obrázek



Co se jinam nevešlo

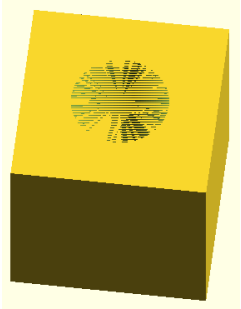
V této kapitole se zaměříme na témata, která nebyla zařazena do předchozích částí. Budeme se věnovat například animacím, asserci, příkazu `render()` a dalším podobným tématům.

Modul `render()`

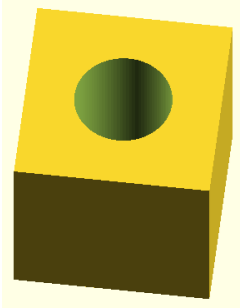
V kapitole OpenSCAD jsme si ukázali, že některé objekty se nevykreslují úplně přesně při použití překladu pomocí `Zobrazit` (F5), viz obrázek 56. Modul `render()` vynutí vyrenderování objektu (nebo jen jeho části).
Viz následující kód, který i při náhledu vykreslí objekt tak, jak je zobrazeno na obrázku 57.

```
render(){
    difference(){
        cube(10, center=true);
        cylinder(d=5,h=10, center=true, $fn=100);
    }
}
```

Obrázek 56: Zobrazit.



Obrázek 57: Zobrazit s `render()`.



Speciální proměnné

Speciální proměnné poskytují alternativní způsob předávání argumentů modulům a funkcím. Všechny uživatelské nebo OpenSCAD definované proměnné začínající `'$'` jsou speciální proměnné. Jak již víme, moduly a funkce vidí i všechny vnější proměnné nejenom ty, které jsou předány jako argumenty nebo jsou definovány uvnitř. U klasických proměnných však platí, že je jim hodnota přiřazena při kompilaci, ne za běhu programu, jsou tedy statické. Speciální proměnné předávají svou hodnotu ze scope, ze kterého je modul nebo funkce volána. To znamená, že speciální proměnné mohou mít potenciálně jinou hodnotu při každém volání modulu nebo funkce. Viz následující příklad. Výpisy, které dostáváme k jednotlivým příkazům jsou v komentáři.

```
promenna  = "normalni";
$specialni = "specialni";

module vypis() echo("promenna: ", promenna,", specialni: ", $specialni);

// vypis vne
echo (promenna, $specialni ); // "normalni", "specialni"

// vypis promenne v cyklu
for ( promenna = [0:1] ){ echo(promenna, $specialni ); vypis();}
// 0, "specialni"
// "promenna: ", "normalni", ", specialni: ", "specialni"
// 1, "specialni"
// "promenna: ", "normalni", ", specialni: ", "specialni"


// cyklus pres specialni promennou
for ( $specialni = [5:6] ){ echo(promenna, $specialni ); vypis();}
// "normalni", 5
// "promenna: ", "normalni", ", specialni: ", 5
// "normalni", 6
// "promenna: ", "normalni", ", specialni: ", 6

// vypis pomoci funkce vypis
vypis();
// "promenna: ", "normalni", ", specialni: ", "specialni"
```

Animace: proměnná `$t`

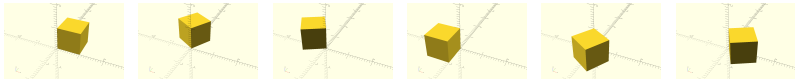
Proměnnou `$t` používáme v příkazech `rotate()` a `translate()` pro animaci, přičemž `$t*360` představuje kompletní cyklus. Pro spuštění animace zvolíme v menu `Window` `Animate`. Pro nastavení animace zadáme hodnoty pro `FPS`[101] a počet kroků `Steps`. Pole `Time` zobrazuje aktuální hodnotu `$t` jako desetinné číslo.

Hodnota `$t` se opakuje od 0 do $(1 - 1/Steps)$.[102] Pro následující kód dostaneme obrázky 58

```
rotate([0,0,$t*360]) cube(1);
```

[101] Frame per second.

[102] Nikdy nedosáhne hodnoty 1, protože by to při rotaci způsobilo „trhání“ animace – dva po sobě jdoucí snímky by byly ve stejném úhlu.



(a) Snímek 1 (b) Snímek 2 (c) Snímek 3 (d) Snímek 4 (e) Snímek 5 (f) Snímek 6

Obrázek 58: Animace.

Úkol 44
Vytvořte animaci tak, aby se objekt (například krychle) pohyboval po eliptické dráze.

Řešení naleznete v kapitole *Řešení vybraných úkolů*.

Animaci můžeme uložit jako sérii obrázků.

Viewport: `$vpr`, `$vpt`, `$vpf` a `$vpd`

Speciální proměnné, které uchovávají informace o kameře.[103] `$vpr` uchovává informaci o otočení kamery, `$vpt` o její poloze a `$vpd` o její vzdálenosti. `$vpf` uchovává informaci o Field of View. Tyto proměnné se dají použít při konstrukci objektů a tím například měnit velikost objektu v závislosti na poloze kamery. Viz následující příklad, kde výška objektu přímo úměrně roste se vzdáleností kamery.

[103] Její poloze vůči souřadnému systému

```
cube([10, 10, $vpd/3]);
```

Případně můžeme hodnoty těchto proměnných měnit a tím měnit polohu kamery. Následující kód rotuje kameru kolem osy z.

```
$vpr = [0, 0, $t * 360];
```

Preview: `$preview`

Speciální proměnná `$preview` má hodnotu `true`, pokud vykreslujeme objekt pomocí `Zobrazit (Preview)`[104], při renderování[105] je rovna `false`.
Následující kód vykreslí kouli v náhledu mnohem méně přesněji, než při renderování.[106]

[104] `F5`
[105] `F6`

[106] Na základě toho, v jakém módu jsme, je nastavena hodnota `$fn`.

```
$fn = $preview ? 12 : 72;
sphere(r = 1);
```

Aserce

Stejně jako celá řada programovacích jazyků i OpenSCAD podporuje aserci. Příkaz `assert()` vyhodnotí podmínku, která mu je předána jako argument `condition`. Pokud se tato podmínka vyhodnotí jako nepravdivá, pak se zastaví generování objektu a v konzoli je zobrazena chybová hláška. To, co je zobrazeno můžeme ovlivnit pomocí nepovinného parametru `message`. Syntaxe je následující.

Průvodce studiem
Aserce je logický pojem označující tvrzení nebo výrok, který se považuje za pravdivý. Často se používá v matematice, logice a informatice. V programování je aserce podmínka, která by měla být za normálních okolností splněna. Pokud není, znamená to chybu v programu, což pomáhá vývojářům odhalit problémy.

```
assert(condition);
assert(condition, message);
```

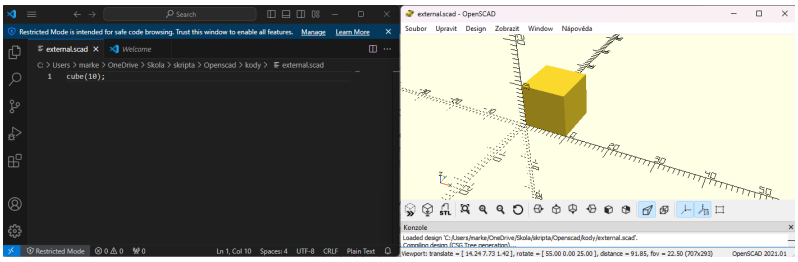
Aserce se využívá zejména při kontrole vstupů modulů. Konkrétní příklad následuje.

```
a = 10;
assert(a < 5); //ERROR: Assertion '(a < 5)' failed in file assert.scad, line 2
assert(a < 5, "Chyba"); //ERROR: Assertion '(a < 5)' failed: "Chyba" in file assert.scad, line 3
```

Externí editory

K psaní kódu v jazyce OpenSCAD není potřeba využívat vývojové prostředí OpenSCAD, ale můžeme zdrojový kód psát v libovolném textovém editoru.[107]
Na obrázku 59 je zdrojový kód psán ve Visual studio code a pomocí prostředí OpenSCAD je jen zobrazován náhled. Pro automatické načítání a vykreslování souboru po jeho otevření v OpenSCAD v záložce `Design` můžeme zaškrtnout možnost `Automaticky načítat a zobrazovat`.

[107] Mnoho editorů dokonce podporuje překlad OpenSCAD jazyka. Například VIM, Atom, Visual studi code a jiné.



Obrázek 59: Kód psaný v externím editoru.

command line
Thingiverse + customizer
Externí editory (nepsaní v openscad)
Další CAD podporující openSCAD
Seznamy
cheatsheet

Řešení vybraných úkolů

Řešení úkolu 7:[108]

```
a = 10;
h = (a/3)*sqrt(6);
R = (sqrt(6)/4)*a;

cylinder(h=h, r1=R, r2 = 0, $fn=3);
```

[108] V řešení jsou použity proměnné pro přehlednost kódu. Jak přesně s proměnnými v OpenSCAD pracujeme si ukážeme později.

Řešení úkolu 8:

```
a = 10;
h = (a/3)*sqrt(6);
v = sqrt((a*a)-((a/2)*(a/2))); // vyska trojuhelnikove postavy
t = (a*sqrt(3))/6; // vzdalenost teziste od hrany

polyhedron(points=[[0,0,0],[0,a,0], [v,a/2,0],[t,a/2,h]],
           faces=[[0,1,2],[0,3,1],[0,2,3],[2,1,3]]);
```

Řešení úkolu 10:

```
fn = 5; // hodnota parametru $fn
pridavek = 1/cos(180/fn);
// predpokladame, ze parametry vyska, polomer jsou definovany drive
cylinder(h=vyska,r=polomer*pridavek,$fn=fn);
```

Řešení úkolu 14:

```
difference(){
    minkowski(){  // telo kostky
        cube(10, center=true);
        sphere(1);
    }
    union(){
        translate([6,0,0]) sphere(1); // 1
        translate([-6,0,0]) // 6
        union(){
            translate([0,-3,0]) sphere(1);
            translate([0, 3,0]) sphere(1);
            translate([0,-3,-3]) sphere(1);
            translate([0, 3,-3]) sphere(1);
            translate([0,-3, 3]) sphere(1);
            translate([0, 3, 3]) sphere(1);
        }
        rotate([0,0,90]) translate([6,0,0]) // 5
        union(){
            sphere(1);
            translate([0,-3,-3]) sphere(1);
            translate([0, 3,-3]) sphere(1);
            translate([0,-3, 3]) sphere(1);
            translate([0, 3, 3]) sphere(1);
        }
        rotate([0,0,-90]) translate([6,0,0]) // 2
        union(){
            translate([0,-3,-3]) sphere(1);
            translate([0, 3, 3]) sphere(1);
        }
        rotate([0,90,0]) translate([6,0,0]) // 3
        union(){
            sphere(1);
            translate([0,-3,-3]) sphere(1);
            translate([0, 3, 3]) sphere(1);
        }
        rotate([0,-90,0]) translate([6,0,0]) // 4
        union(){
            translate([0,-3,-3]) sphere(1);
            translate([0, 3,-3]) sphere(1);
            translate([0,-3, 3]) sphere(1);
            translate([0, 3, 3]) sphere(1);
        }
    }
}
```

Řešení úkolu 17:

```
$fn = 100;
difference(){
    offset(r=2) square(5, center=true);
    offset(r=1) square(5, center=true);
}
```

Řešení úkolu 19:

```
rotate_extrude()
difference(){
    circle(r=10);
    translate([-10,0,0]) square(20,center=true);
}
```

Řešení úkolu 22:

```
$fn= 100;
difference(){
    minkowski(){  // telo kostky
        cube(12, center=true);
        sphere(1);
    }
    union(){
        // 1
        translate([0,0,6]) linear_extrude(1)
            text("1", valign="center", halign="center", font="Arial");
        // 6
        rotate([0,180,0]) translate([0,0,6]) linear_extrude(1)
            text("6", valign="center", halign="center", font="Arial");
        // 5
        rotate([0,90,0]) translate([0,0,6]) linear_extrude(1)
            text("5", valign="center", halign="center", font="Arial");
        // 2
        rotate([0,-90,0]) translate([0,0,6]) linear_extrude(1)
            text("2", valign="center", halign="center", font="Arial");
        // 4
        rotate([90,0,0]) translate([0,0,6]) linear_extrude(1)
            text("4", valign="center", halign="center", font="Arial");
        // 3
        rotate([-90,0,0]) translate([0,0,6]) linear_extrude(1)
            text("3", valign="center", halign="center", font="Arial");

    }
}
```

Řešení úkolu 24:

```
translate([0, 60, 0])
    rotate_extrude(angle=270)
        translate([40, 0])
            circle(10);

translate([60, 0, 0])
    rotate([0,0,180])
        rotate_extrude(angle=270)
            translate([40, 0])
                circle(10);

rotate_extrude(angle=90)
    translate([20, 0])
        circle(10);

translate([60,60,0])
    rotate([0,0,180])
        rotate_extrude(angle=90)
            translate([20, 0])
                circle(10);
```

Řešení úkolu 36:

```
n = 4;
a = 5; // velikost krychle

for(i = [0 : n - 1], j = [0 : n - 1]){
    translate([i*a, j*a, 0]) cube([a, a, a*(n-(i + j))]);
}
```

Řešení úkolu 40:

```
function delka(text, pocet = 0) =
    text[pocet] == undef ? pocet : delka(text, pocet + 1);

echo(delka("test"));  // ECHO: 4
```

Řešení úkolu 44:

```
a = 5;
b = 10;
```

```
translate([a*sin($t*360), b*cos($t*360)])
cube(1, center=true);
```

Doporučená literatura

[1] https://openscad.org/documentation.html

[2] https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Mathematical_Functions

[3] https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/String_Functions

[4] https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Type_Test_Functions

[5] Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Felkel: Moderní počítačová grafika. Computer Press, 2004.