INFORMATIKA

Dělení kostky na vlastní oči

ŠÁRKA GERGELITSOVÁ – TOMÁŠ HOLAN

Matematicko-fyzikální fakulta UK, Praha

V [1] jsme před časem popsali úlohu, kolika způsoby lze rozdělit kostku, složenou z $2 \times 2 \times 2$ kostiček, na dvě souvislé části. Někteří z čtenářů úlohu vyřešili sami, jiní možná uvěřili našemu řešení [2], ale nebylo by lepší, kdybychom se o tom mohli přesvědčit na vlastní oči?

Obrázek často vydá za mnoho slov, říká se. A to se říká o dvourozměrných obrázcích, ale do třírozměrného prostoru se toho vejde víc než na dvourozměrný papír. Mohli bychom zobrazit všechny možnosti rozdělení krychle na části v trojrozměrném prostoru tak, aby si je každý mohl prohlédnout? Netrpěliví čtenáři se mohou podívat sem: https://projektor.geometry.cz/?co=show&did=335, ti trpěliví nechť čtou dále.

Možnosti trojrozměrného zobrazení

Když jsme (my, autoři tohoto článku) chodili do školy, mívali jsme v kabinetech zeměpisu, přírodopisu, ale i fyziky nebo chemie, řadu trojrozměrných modelů vyrobených ze dřeva a lepenky. Dnes je snazší si takový model vyrobit a zobrazit v počítači, ať už se na něj budeme dívat pomocí brýlí pro virtuální realitu nebo jen jako na obrázek, kterým můžeme různě pohybovat a otáčet.

Pokud chceme vytvořit trojrozměrný model, můžeme použít některý z řady modelovacích programů (Blender, 3DSMax, ...), ale pokud nechceme nic instalovat a pokud se nechceme učit ovládat nový program, stačí nám nástroje a jazyky, které dokážou trojrozměrné objekty popsat textem a následně zobrazit ve webovém prohlížeči. Takovým jazykem je VRML [4], dnes nahrazované svým následníkem X3D [5] a X3DOM [6]. Trojrozměrné objekty se tam popisují pomocí zvláštní syntaxe a není těžké

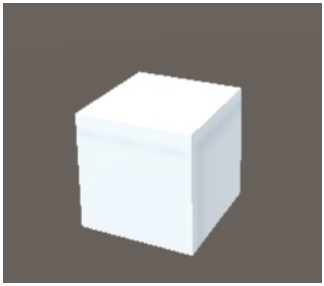
pomocí několika řádků definovat trojrozměrnou scénu, kterou potom prohlížeč dokáže zobrazit.

Nová syntaxe

Protože jsme se ale nechtěli zatěžovat syntaxí VRML ani X3D, vymysleli jsme si ještě jednodušší jazyk a vytvořili jsme k němu prohlížeč, to všechno v jedné webové stránce [7]. Scéna se v něm popisuje jako text, každý řádek odpovídá jednomu objektu a když napíšeme třeba

```
C
```

dostaneme krychli:

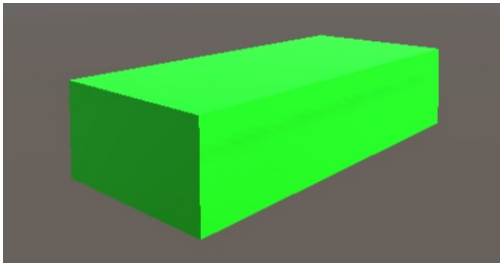


Podobně příkaz `S` vytvoří kouli nebo `Y` válec.

Když nejsme spokojeni s velikostí, můžeme do stejného řádku přidat parametry ovlivňující velikost (S), ale také rotaci (R), barvu (C) nebo polohu (P), takže třeba příkaz

```
C    S 2 0.5 1    C 0 1 0    P 0 1 0
```

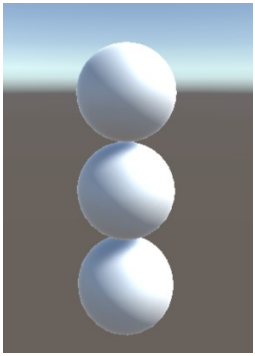
vytvoří zelenou cihlu posunutou o 1 vzhůru (krychle škálovaná v ose x na dvojnásobek, v ose y na polovinu, jejíž barva je vyjádřena pomocí RGB složek $(0, 1, 0)$ a která je posunutá z počátku o vektor $(0, 1, 0)$).



Příkazy

```
S
S  P 0 1 0
S  P 0 2 0
```

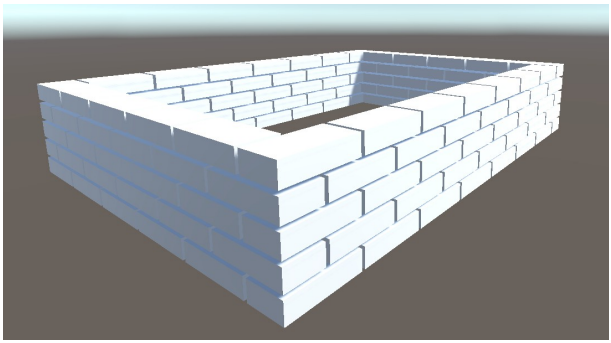
vyrobí sněhuláka:



Program, který píše program

Nepíšeme tenhle článek proto, abychom ukazovali další z mnoha možností, jak popisovat trojrozměrné konstrukce, ani nezapomínáme na to, čím jsme začali, tedy dělením kostky, už se k tomu blížíme.

Takže máme jazyk, kterým můžeme popsat trojrozměrnou scénu, a máme program, který tuto scénu potom dokáže zobrazit. Další krok, který chceme představit, je, že ten popis scény nemusíme psát vlastnoručně, ale můžeme si napsat program, který ho napíše za nás. Anebo se na to můžeme podívat z druhé strany a představit si, že až budeme příště v hodinách informatiky učit cyklus, tak místo úlohy *napište program, který vypíše čísla od 1 do 10* můžeme řešit úlohu *napište program, který postaví zeď*. Zeď přitom může pro různě pokročilé žáky znamenat různé věci, od řady kostek umístěných ve vodorovné i svislé ose až po opravdovou zeď se správně propojenými cihlami:



Jak je to s dělením kostky?

Napíšeme si program, který projde všechny možnosti, jak lze osm kostiček, ze kterých se skládá krychle $2\times2\times2$, rozdělit do dvou množin. Protože je kostek osm a každá bude patřit do první nebo do druhé množiny, bude takových dělení existovat 2^8, tedy 256 možností. Tyto možnosti můžeme očíslovat čísly 0..255 a když si představíme takové číslo zapsané ve dvojkové soustavě, potom číslice 0, resp. 1, na i-té pozici bude znamenat, že i-tá kostička patří do první, resp. druhé množiny.

Pro každou z těchto možností program spočte, na kolik kusů se takto rozdělená kostka rozpadne, a pokud ty kusy budou právě dva, tak je to platné rozdělení a obarví jednu část modře a druhou zeleně; pokud bude počet kusů jiný, tak obě části rozdělené kostky obarví jinými dvěma barvami.

Pro zjištění počtu kusů použijeme následující algoritmus: Na začátku jednotlivým kostičkám přiřadíme čísla 0, 1, 2, ..., 7 určující, do které *komponenty souvislosti* ta která kostička patří. Potom se budeme dívat na dvojice kostiček, které spolu v původní krychli $2\times2\times2$ *sousedily*, a pokud obě kostičky patří do stejné množiny (mají v binárním zápisu na své pozici obě nulu nebo obě jedničku), spojíme tyto komponenty souvislosti tak, že všem kostičkám, které mají číslo komponenty shodné s číslem komponenty druhé kostičky, nastavíme číslo komponenty, které má první kostička.

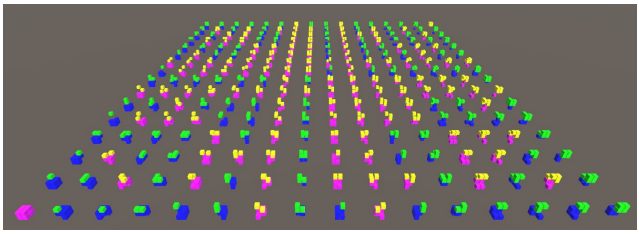
Zdrojový kód vypadá takto:

```
N = 2
KOSTEK = N**3
DELENI = 2**KOSTEK
ROZMER = 2**(KOSTEK/2)
K = 1       # rozměr kostičky
Km = 1.1    # rozestupy kostiček
S = N*Km   # rozměr jednoho dělení
MEZERA = 3
Sm = (1+MEZERA)*S # rozestupy dělení

def PocetKomponent( deleni ):
    # binární zápis dělení určuje,
    # do které části kostička patří
    # správné dělení by mělo mít dvě souvislé komponenty
    bd = ""
    for _ in range(KOSTEK):
```

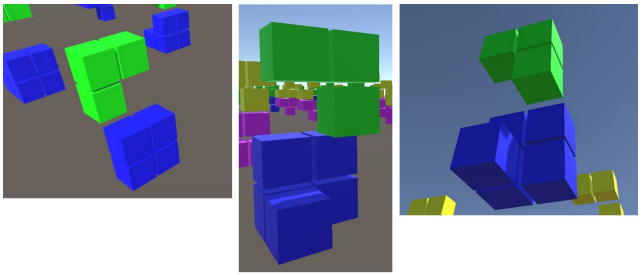
```
        bd = bd+"01"[deleni % 2]
        deleni = deleni//2
    HRANY2x2x2 = [ (0,1), (0,2), (1,3), (2,3),
    # které kostičky spolu sousedí
                   (4,5), (4,6), (5,7), (6,7),
                   (0,4), (1,5), (2,6), (3,7) ]
    komp = [i for i in range(KOSTEK)]
    for h in HRANY2x2x2:
        if bd[ h[0] ]==bd[ h[1] ]:
        #jsou ve stejné části kostky
            if komp[h[1]]!=komp[h[0]]:
            # jsou zatím v různé komponentě...
                zceho, naco =komp[h[1]], komp[h[0]]
                for i in range(KOSTEK):
                    if komp[i]==zceho:
                        komp[i]=naco
                    # ...převedeme do stejné komponenty
    return len(set(komp))

for d in range(DELENI):
    dx = d % ROZMER
    dz = d // ROZMER

    # jedno dělení:
    pocet = PocetKomponent( d )
    dd = d # pro převod do dvojkové soustavy
    for i in range(KOSTEK):
        x = i % N
        y = (i // N) % N
        z = i // (N*N)
        if dd%2==0:
            barva = "0 0 1"
            Y = 0
        else:
            barva = "0 1 0"
            Y = 1 * S
        if pocet!=2:
            barva = "1"+barva[1:]
```

```
dd = dd // 2
print( f"C  S {K} {K} {K}  P {Sm*dx+x*Km} {0+y*Km +Y}
↪  {Sm*dz+z*Km}   C {barva}\n" )
```

Výsledný text, který může být obtížné číst, protože těch kostek je hodně, potom nakopírujeme do vstupního pole výše zmíněného prohlížeče a můžeme prohlížet možnosti dělení kostky. Všechny najednou...



...i každou zblízka a z různých stran:



Další podobné úlohy?

Podobně můžeme postupovat i v případě jiných úloh. Například u otázky: *Kolik různých souvislých útvarů lze slepit z jedné, dvou, tří, čtyř, pěti, ... kostek?*

Nějaký vzoreček nebo návod, který by to řešil obecně, neznáme. Ale není těžké napsat program, který bude *prohledávat* všechny možnosti. Můžeme si pamatovat *seznam obsazených* polí a k tomu také *seznam sousedních* polí, abychom je nemuseli pokaždé hledat. Také se nám bude hodit pamatovat si, jaké možnosti/tvary už jsme našli, abychom je nezapočítávali víckrát.

Program by potom mohl vypadat třeba takto:

```
# kolik různých souvislých tvarů  jde poskládat ze (7)
↪  kostiček?

# tvar = (počet,obsazené,sousední),
# obsazené i sousední jsou množiny (x,y,z)
# základní tvar:
# - ze všech možných pootočení si nechat jenom jedno,
# - zároveň posouvat k počátku ve všech osách
# - vybírat to, které má nejmenší (textově) popis

# BFS:
fronta = [ (0,set(),{(0,0,0)}) ]
zname = set()  # známé "obsazené"

def sousedi( x,y,z ):
    return { (x-1,y,z), (x+1,y,z), (x,y-1,z), (x,y+1,z),
    ↪  (x,y,z-1), (x,y,z+1) }

vysledny_seznam = []
KOSTKY = 7
kolik = 0
last_pocet = 0
while len(fronta) > 0:
    tvar = fronta.pop(0)
    pocet, obsazene, sousedni = tvar
    if pocet > last_pocet:
        print( f"{last_pocet}: {kolik}" )
        kolik = 0
        last_pocet = pocet
    kolik += 1

    if pocet==KOSTKY:
        vysledny_seznam.append(zakladniTvar(obsazene)[1])
        # [1] = rotace a posunutá k počátku
    elif pocet > KOSTKY:
        break

    for (x,y,z) in sousedni:
```

```
            new_obsazene = obsazene.union( set({(x,y,z)}) )
            zakl = zakladniTvar( new_obsazene )[0]
            # [0] = popis obsazených
            if zakl in zname:  # tenhle tvar už mám
                continue
            zname.add( zakl )  # aby se příště neopakoval
            # je to NOVÝ tvar => doplnit jeho další atributy
            new_sousedni = sousedni.union( sousedi(x, y, z) )
            new_sousedni = new_sousedni.difference(new_obsazene)
            fronta.append( (pocet+1, new_obsazene, new_sousedni)
            ↪  )

# vypsat výsledný seznam:
K = 1
Km = 1.1
BLOK = 10
RADKA = int(len(vysledny_seznam)**(1/2))
pocet = 0

f = open("slepenec.txt",'w')
for obsazene in vysledny_seznam:
    x0 = BLOK*(pocet % RADKA)
    z0 = BLOK*(pocet // RADKA)
    pocet += 1
    for (x,y,z) in obsazene:
        f.write( f"C  P {Km*(x0+x)} {Km*y} {Km*(z0+z)}   C 0
        ↪  1 0\n" )
f.close()
```

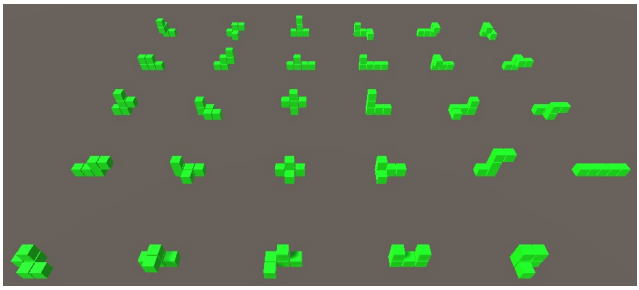
Asi nejobtížnější na tomto programu je (vynechaná) funkce `zakladni Tvar(new_obsazene)`, která množinu obsazených pozic převede vždycky na stejný tvar bez ohledu na rotaci nebo posunutí, abychom mohli porovnávat, jestli jsme tuhle kombinaci kostek už někdy viděli.

Počty možností pro rostoucí počty kostek jsou:

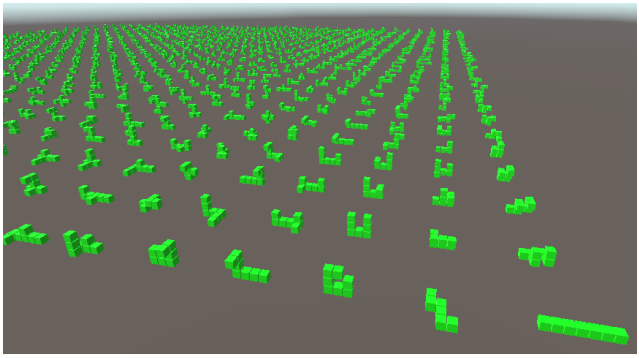
```
0: 1
1: 1
2: 1
3: 2
```

```
4: 8
5: 29
6: 166
7: 1023
...
```

a výsledné obrázky mohou vypadat třeba takhle pro pět kostek [7]:



a takhle pro sedm kostek [8]:



Závěr

Obrázek nám pomůže vidět něco, co bychom si jinak mohli jenom představovat. Prostorový obrázek může být ještě o něco názornější než dvourozměrný obrázek. A vytvořit prostorový obrázek může být docela snadné, pokud máme jednoduchý jazyk pro jeho popis, a ještě snazší, pokud pro vytváření použijeme program. Zkuste to také!

Literatura

[1] Zajímavé matematické úlohy, Úloha 260. Matematika–fyzika–informatika, roč. 29 (2020), č. 1, s. 21.

[2] Zajímavé matematické úlohy, Úloha 260. Matematika-fyzika–informatika, roč. 29 (2020), č. 3, s. 198–200.

[3] VRML Virtual Reality Modeling Language. [online] 1995 cit. [2023-11-01]. Dostupné z: https://www.w3.org/MarkUp/VRML/

[4] The Web3D Consortium. [online] 2023 cit. [2023-11-01]. Dostupné z: https://www.web3d.org/

[5] Official x3dom documentation. [online] 1995 cit. [2023-11-01]. Dostupné z: https://doc.x3dom.org/

[6] Projektor. [online] 2023 cit. [2023-11-01]. Dostupné z: https://projektor.geometry.cz/

[7] Slepenec 5. [online] 2023 cit. [2023-11-01]. Dostupné z: https://projektor.geometry.cz/?co=show&did=358

[8] Slepenec 7. [online] 2023 cit. [2023-11-01]. Dostupné z: https://projektor.geometry.cz/?co=show&did=359

Počítačová grafika, 4. díl

EDUARD BARTL

Přírodovědecká fakulta UP, Olomouc

Článek volně navazuje na předchozí tři díly, zabývá se rasterizací základních geometrických objektů. Podrobně se věnuje rasterizaci úsečky pomocí algoritmu DDA. Může sloužit jako pomůcka pro středoškolské učitele informatiky a výpočetní techniky.

Rasterizací objektů rozumíme kreslení těchto objektů do rastru zobrazovacího zařízení, v našem případě se jedná o displej. Budeme mít k dispozici matematický popis nějakého základního geometrického objektu, například úsečky. Tento objekt je tvořen nekonečným množstvím bezrozměrných geometrických bodů (viz také první díl tohoto seriálu [1]) a naším úkolem je