UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATICKÉ ANALÝZY A APLIKACÍ MATEMATIKY

BAKALÁŘSKÁ PRÁCE

Obrázkový font v Metafontu

Vedoucí bakalářské práce:
RNDr. Miloslav Závodný
Rok odevzdání: 2014

Vypracoval:
Michaela Brenkusová
M-E bankovnictví, III. ročník

Prohlášení

Prohlašuji, že jsem vytvořila tuto bakalářskou práci samostatně za vedení RNDr. Miloslava Závodného a že jsem v seznamu použité literatury uvedla všechny zdroje použité při zpracování práce.

V Olomouci dne 10. dubna 2014

Poděkování

Ráda bych na tomto místě poděkovala vedoucímu bakalářské práce RNDr. Miloslavovi Závodnému za obětavou spolupráci i za čas, který mi věnoval při konzultacích.

Obsah

Úvod **4**

1 Program TEX **5**
1.1 Donald Knuth 5
1.2 Jak program pracuje 7

2 METAFONT **8**
2.1 Jak program pracuje 8
2.2 Jednoduché obrázky 10
2.3 Geometrické tvary 12
2.3.1 Trojúhelník 12
2.3.2 Čtverec 13
2.3.3 Obdélník 14
2.3.4 Kružnice 14
2.4 Další obrázky 15
2.5 Proměnná picture 26

3 Kombinatorické struktury **28**
3.1 Sébastien Truchet 28
3.2 Zdeněk Sýkora 32

Závěr **37**

Literatura **38**

Úvod

Cílem bakalářské práce je ukázat možnosti programu METAFONT při tvorbě obrázkových fontů. Nabízené možnosti pak předvést na fontu, pomocí něhož bude možné sestavit kombinatorický obraz na základě již existujícího vzoru.

Při výběru tématu bakalářské práce jsem si svojí volbou nebyla zcela jistá. Měla jsem obavu z náročnosti programů TEXa METAFONT, ale s postupem práce jsem zjistila, jak dobrá moje volba byla. Předtím jsem pracovala pouze s programy firmy Microsoft, především Wordem, v němž je zadávání sazby matematických výrazů složité. Za existenci programů TEX i METAFONT vděčíme Donaldu Ervinu Knuthovi.

V úvodu práce se stručně zmíním o typografickém systému TEX, do kterého vytvořený obrázkový font zařadíme a pomocí kterého vytvoříme kombinatorický obraz. V další části vysvětlím způsob práce s programem METAFONT. A nakonec se pokusím vytvořit zmíněný font pro sestavení kombinatorického obrazu.

1. Program TEX

TEX je typografický systém, který si můžeme zdarma stáhnout ze stránek TEX Users Group (TUG) `http://www.tug.org/`. TUG je sdružení uživatelů TEXu, které vzniklo v USA. I u nás existuje takové sdružení. Nazývá se Československé sdružení uživatelů TEXu (zkráceně CSTUG), členství v něm je dobrovolné, poplatky za něj se ovšem platí. Toto sdružení se zabývá především šířením TEXu v Čechách a na Slovensku, tj. i podporou sazby v českém a slovenském jazyce.

TEX je v akademickém prostředí velice rozšířený program, a to především mezi matematiky a informatiky. Používají ho ale mnohé matematické časopisy i obyčejní nadšenci pro dobrou sazbu. TEX je totiž určen pro sazbu krásných knih, zvláště těch, které obsahují spoustu matematických formulí.

Abych mohla s TEXem pracovat, musela jsem nejdříve zvolit některou jeho distribuci a nainstalovat ji na svůj počítač – i to byl pro mě zpočátku problém, protože uživatel musí vědět, co všechno bude potřebovat, alternativou je instalace všeho, tj. např. i podpora sazby v korejštině.

Dalším problémem pro uživatele Wordu je to, že TEX je značkovací systém, zatímco Word je WYSIWYG (What You See Is What You Get) textový editor – uživatel pomocí myši a nabídek menu umisťuje objekty na zvolené místo na stránce, vidí tedy, co udělal. V TEXu jsou formátovací značky zapisovány přímo do textu, sazeč přímo nevidí výsledný efekt.

1.1. Donald Knuth

Donald Ervin Knuth (DEK) se narodil 10. ledna 1938 ve městě Milwaukee, stát Wisconsin, USA. Vyrůstal jako syn učitele účetnictví, po kterém zdědil touhu po vzdělání. Nevěnoval se ale od malička matematice, nýbrž hudbě. Hrál na dva hudební nástroje, byl skladatelem. Hudbě se věnoval celou střední školu. Později ale dostal návrh, studovat fyziku na Clevelandské univerzitě, kam opravdu nakonec nastoupil. Netrvalo však dlouho a Donald se rozhodl přestoupit na jiný obor – matematickou analýzu. V roce 1956 se Knuth seznámil se svým prvním počítačem a podlehl jeho kouzlu.

Donald Ervin Knuth byl průkopníkem informatiky – v roce 1960 obdržel zároveň bakalářský i magisterský titul v oboru matematika a v roce 1963 titul doktorský.

Monografii „Umění programovat“ psal od roku 1962. Čtyři díly již napsal, na další dva se chystá. Sám tvrdí, že monografii dokončí v roce 2020. Všechny tyto knihy jsou velice ceněné stejně jako sám Donald Knuth.

Při psaní třetí knihy cyklu „Umění programovat“ začal vytvářet svůj typografický systém TEXa to z jednoho prostého důvodu – když uviděl tehdejší počítačově vysázenou předlohu své knihy, byl rozhořčen její nedokonalostí, rozhodl se tedy vytvořit typografický systém sám. Svůj program TEX zdokonaluje od roku 1977 dodnes. Verze TEXu nejsou číslovány klasicky po řadě, ale podle rozvoje Ludolfova čísla (nyní máme verzi 3.1415926). Knuth chce, aby jeho dílo zůstalo jeho – aby po jeho smrti nikdo TEX nezdokonaloval, aby zůstal takový, jaký ho zanechá.

DEK je informatikem, matematikem, nyní již emeritním profesorem na Stanfordově univerzitě. Zasloužil se o pokrok v programování díky své knize „Umění programovat“. Vedlejším efektem vzniku této knihy bylo vytvoření programu TEX (a jeho součásti METAFONT), jak bylo již uvedeno.

Za svoji práci získal několik cen [3]:

- V roce 1974 –Turingova cena
- V roce 1979 – Národní vyznamenání za vědu (předával prezident Carter)
- V roce 1995 – Medaile Johna von Neumanna
- V roce 1996 – Kvótská cena
- V roce 1996 – titul dr. h. c. Masarykovy univerzity

K zásadním knihám Donalda Ervina Knutha patří [3]:

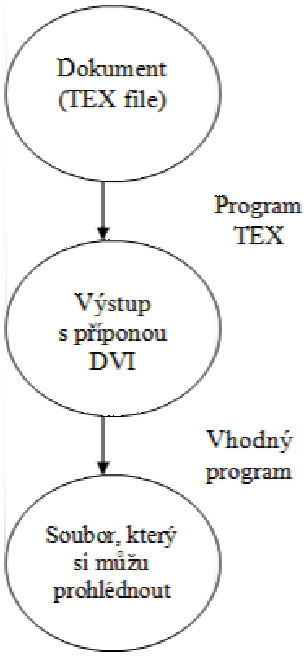
- Umění programování (4 svazky)
- The TeXbook
- The METAFONTbook

1.2. Jak program pracuje

Nejprve editorem připravíme soubor obsahující zpracovávaný text a značky určující vzhled sazby, tzv. dokument (TEX file). Tento dokument zpracujeme programem TEX se zvoleným formátem – výstupem je soubor s příponou dvi. Tento soubor obsahuje abstraktní popis vzhledu sazby navržený Knuthem, text je zde reprezentován tzv. boxy, pravoúhelníky se zadanou šířkou, výškou a hloubkou. Na použité fonty je v boxech pouze odkazováno, nejsou v dvi souboru zařazeny.

Dvi soubor tedy popisuje rozložení boxů na stránce, neobsahuje znaky písma (a jiné objekty) s nimi spřažené – popis je nezávislý na výstupním zařízení (DeVice Independent). Nakonec pomocí vhodného programu obsluhujícího výstupní zařízení (tzv. ovladače) propojíme odkazy dvi souboru s konkrétními tvary písmen. Výsledek si můžeme prohlédnout.

V současnosti se dvi popis převádí ovladačem dvips do postskriptu nebo ovladačem dvipdfm do formátu pdf (portable document file). Existuje rovněž verze pdfTEX pro přímý výstup dokumentu do pdf. Dále se pak již pracuje s postskriptem nebo pdf souborem.



2. METAFONT

Donald Knuth rovněž sestavil program METAFONT, který slouží pro programování a digitalizaci písem (fontů) potřebných pro sazbu. První verzi METAFONTu uveřejnil Donald Knuth v roce 1977. Tato verze měla nedostatky, proto Knuth vyvinul zcela novou verzi v roce 1984. Nyní pracujeme s verzí 2.718281 (číslovanou podle rozvoje Eulerova čísla).

METAFONT je program, který zpracovává textový soubor (zdrojový soubor fontu), v němž se znaky fontu definují pomocí formulí analytické geometrie. Byl navržený jako doplněk sázecího systému TEX, je-li to potřebné, volá ho TEX automaticky, aby vygeneroval dva soubory fontu nutné pro práci TEXu – soubor obsahující údaje o boxech (metriku fontu, font metrics) a soubor obsahující údaj o vzhledu jednotlivých znaků v podobě bitové mapy (generický font, generic font) – ze zdrojového souboru, pokud existuje. Lze ho volat z příkazového řádku příkazem `mf`, povinným parametrem je jméno zdrojového souboru.

Pro vlastní práci potřebuje TEX pouze metriku, chybí-li nám ale bitová mapa fontu, výsledek neuvidíme. Můžeme ale vysázet text ve fontu, který má tiskárna, a ta dvi soubor zpracuje sama. Koncepce práce s fonty navržená Knuthem umožňuje použít libovolný formát fontu, např. TTF (True Type Font).

2.1. Jak program pracuje

Nejprve napíšeme v programovacím jazyku METAFONTu soubor s našimi požadavky (zdrojový soubor, source file). Přípona tohoto souboru se obvykle volí mf, pak ji nemusíme při volání programu uvádět. Nechť se náš soubor jmenuje `pokus.mf`.

Soubor necháme zpracovat METAFONTem, tj. zavoláme např. z příkazové řádky `mf pokus`. Pokud jsme neudělali chybu, získáme dva soubory, které jsou sice stejně pojmenované, ale s rozdílnou příponou – zmíněnou metriku `pokus.tfm` a generický font `pokus.<číslo>gf`. Soubor `pokus.<číslo>gf` bývá komprimován programem gftopk, pak má příponu .<číslo>pk – `pokus.<číslo>pk`. Zde `<číslo>` udává rozlišení, ve kterém je bitová mapa vyrobena, to je dáno tzv.

módem, v němž METAFONT pracuje. Mód může být nastaven ve zdrojovém souboru, nebo být zadán jako parametr při volání programu `mf`. Mód nastavený pro automatické generování metriky a bitové mapy je `ljfour` a výsledkem je bitová mapa v rozlišení 600 dpi, pak máme `pokus.600gf`, resp. `pokus.600pk`. Módy jsou definovány v souboru `modes.mf` v adresáři `\texmf-dist\metafont\misc`.

Jak již bylo řečeno, TEX potřebuje tfm soubor, ovladač výstupního zařízení pk soubor.

Zde je nutné poznamenat, že způsob pojmenování souboru s bitovou mapou se liší podle distribuce TEXu. Uvedený způsob je novější a používá ho distribuce TEXLive, fonty stejné rodiny jsou v jednom adresáři, různé rozlišení se liší číslem v příponě. Distribuce MikTEX pracuje s původně navrženým způsobem, bitové mapy nemají číslo, tj. jmenují se `pokus.pk`[1] a jednotlivá různá rozlišení jsou v podadresářích, jejichž jméno je dáno rozlišením bitové mapy, např . `600DPI`.

METAFONT je také programovací jazyk – abeceda jazyka, jeho syntaxe i sémantika je popsána v Knuthově METAFONTbooku.

Ve zdrojovém souboru fontu je pro každý znak definován box s ním spřažený (pro tfm). Je tedy definována jeho výška, šířka, hloubka a také počátek (origin). Počátek (boxu) je bod, který je při usazení boxu na účaří umístěn do aktuálního bodu sazby, tj. na místo na stránce, kam se má právě sázet (hloubka a výška boxu se vztahují k účaří). Počátek boxu je rovněž počátkem souřadnicové soustavy, k němuž se vztahují souřadnice bodů definujících znak fontu.

Jednotka soustavy souřadnic je definována v konstantě `u`. Nejprve se `u` definuje v „reálných“ jednotkách (`u#:=1mm#;`), poté se přepočte na pixely výstupního zařízení (`define_pixels(u);`), jehož mód musí být nejprve nastaven (`mode_setup;`).

Dalšími příkazy definujeme v této souřadnicové soustavě body, které spojujeme různými druhy čar zvoleným typem kreslícího pera. Definujeme vektory, sčítáme je a přičítáme k bodům, počítáme úhly, které vektory svírají atd. Metafont je také schopen řešit rovnici o jedné neznámé `whatever`, provádět geometrické

[1] Ze jména souboru nelze v tomto případě určit rozlišení bitové mapy, což může vyvolat problémy při předávání dokumentu s tímto fontem jinému uživateli. Zařízení, na kterém bude tisknout, může mít jiné rozlišení a výsledek bude jen těžko použitelný.

transformace, provádět aritmetické operace, vyhodnocovat některé elementární funkce, generovat čísla z rovnoměrného i normálního rozdělení aj.

příklad	příkaz
součet	$x + y$
rozdíl	$x - y$
součin	$x * y$
podíl	x/y
$\sin x$	$\mathrm{sind}\, x$ (ve stupňích)
$\cos x$	$\mathrm{cosd}\, x$ (ve stupňích)
normální rozdělení	normaldeviate
rovnoměrné rozdělení	uniformdeviate

Protože je znak písma de facto obrázek, program METAFONT je možné použít k programování obrázků. Této možnosti se budeme dále věnovat. Přitom ale nepotřebujeme ovládat všechny možnosti, které METAFONT nabízí, stačí nám jen ty, které se týkají čárové grafiky, tj. ty, které budeme potřebovat k naprogramování obrázkového fontu.

2.2. Jednoduché obrázky

V poznámkovém bloku vytvoříme obrázek (viz. níže) s příponou `mf` (např. obrázek.mf). Do příkazového řádku zadáme příkaz `mf` a název našeho obrázku (obrázek) a vytvoří se nám dva soubory. První je soubor s názvem `obrázek.<číslo>gf` a druhý s názvem `obrázek.log`. Poté vložíme obrázek do našeho dokumentu následujícím způsobem:

```
\font\obra=obrázek
```

Vytvoří se další soubory –`obrázek.<číslo>pk` a `obrázek.tfm`. Pokud uděláme změnu v souboru `obrázek.mf`, musíme vždy tyto dva soubory smazat, aby se nový obrázek (po změně) ukázal v našem dokumentu změněný.

```
mode_setup;
u#:=1pt#;
define_pixels(u);
input osy;
beginchar(2,18u#,0,0);
z1=(2u,0); z2=(w,0);
pickup pencircle scaled 0.4pt;
draw z1--z2;
endchar;
end;
```



```
mode_setup;
u#:=1pt#;
define_pixels(u);
input osy;
beginchar(1,18u#,0,0);
z1=(2u,0); z2=(w,0);
pickup pencircle scaled 0.4pt;
draw z1--z2;
filldraw arrow(3.5u,0) shifted z2;
endchar;
end;
```



```
mode_setup;
u#:=1pt#;
define_pixels(u);
input osy;
beginchar(1,18u#,0,0);
z1=(2u,0); z2=(w,0);
pickup pencircle scaled 0.4pt;
draw z1--z2;
```

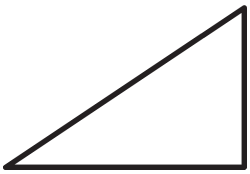
```
filldraw arrow(3.5u,0) shifted z2;
endchar;
beginchar(2,18u#,0,0);
z1=(2u,0); z2=(w,0);
pickup pencircle scaled 0.4pt;
draw z1--z2;
endchar;
beginchar(3,18u#,0,0);
z1=(2u,0); z2=(w,0);
pickup pencircle scaled 0.4pt;
draw z1--z2;
filldraw arrow(3.5u,180) shifted z1;
endchar;
end
```

2.3. Geometrické tvary

Pro jednodušší práci s METAFONTem budu používat příkaz `screenstrokes`, který mi s postupem mé práce přišel velice důležitý. Tento příkaz nám totiž poskytne náhled obrázku již po prvním přeložení.

2.3.1. Trojúhelník

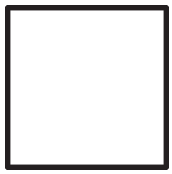
Pokud chceme nakreslit trojúhelník, stačí zadat tři body `z1,z2,z3`. Dále zvolíme z několika možností typů pera (např. pencircle, pensquare nebo penrazor). Zadáme příkaz `draw`, který nám propojí zadané body.



```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(30u,0);z3=(30u,20u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z1;
endchar;
end
```

2.3.2. Čtverec

Stejně jako u trojúhelníku zadáme body, v našem případě čtyři `z1,z2,z3,z4`. Musíme správně nadefinovat tyto body, aby bod `z1` a bod `z2` byly od sebe stejně vzdálené jako body `z3,z4`, `z1,z4` a `z2,z3`, tedy abychom vytvořili opravdu čtverec. Další kroky jsou shodné jako v případě trojúhelníku – zvolíme typ pera a příkazem `draw` propojíme námi nadefinované body.



```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(20u,0);z3=(20u,20u); z4=(0,20u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z4--z1;
endchar;
end
```

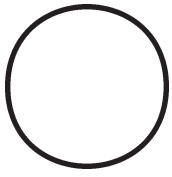
2.3.3. Obdélník

Zadáme si body `z1,z2,z3,z4` a jako v případě čtverce, musíme správně nadefinovat vzdálenosti mezi jednotlivými body, aby vznikl obdélník. Další postup je stejný jako u dvou předešlých.

```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(30u,0);z3=(30u,15u); z4=(0,15u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z4--z1;
endchar;
end
```

2.3.4. Kružnice

V případě kružnice nám stačí zadat pouze dva body `z1`, `z2`, které budou mít stejnou x-ovou hodnotu. Jediný rozdíl oproti předešlým obrázkům, kde jsme body spojovaly pomocí přímek je, že v tomto případě použijeme příkaz, který nám tyto dva body spojí křivkou, která bude mít tvar kružnice.



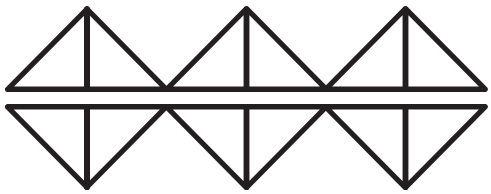
```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(20u,0);
pickup pencircle scaled 2pt;
draw z1..z2..z1;
endchar;
end
```

2.4. Další obrázky

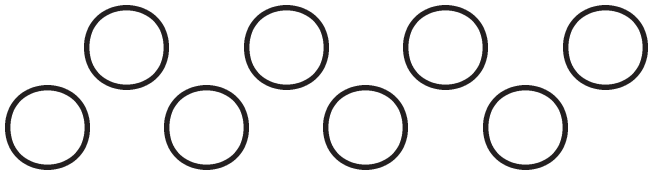
```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(30u,0);z3=(30u,30u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z1;
endchar;
beginchar(2,30u#,30u#,0);
z1=(0,0); z2=(30u,0);z3=(30u,30u); z4=(0,30u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z4--z1;
endchar;
beginchar(3,30u#,30u#,0);
z1=(0,0); z2=(30u,0);z3=(0,30u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z1;
endchar;
end
```

```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(10u,0);z3=(10u,10u); z4=(0,10u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z4--z1;
endchar;
beginchar(2,30u#,30u#,0);
z1=(0,0); z2=(5u,0);z3=(10u,0u);
z4=(10u,5u); z5=(10u,10u); z6=(5u,10u);
z7=(0,10u); z8=(0,5u);
pickup pencircle scaled 2pt;
draw z1--z5;
draw z3--z7;
draw z8--z4;
draw z6--z2;
endchar;
beginchar(3,30u#,30u#,0);
z1=(0,0); z2=(5u,0);z3=(10u,0u);
z4=(10u,5u); z5=(10u,10u); z6=(5u,10u);
z7=(0,10u); z8=(0,5u);
pickup pencircle scaled 2pt;
draw z1--z5;
draw z3--z7;
draw z8--z4;
draw z6--z2;
```

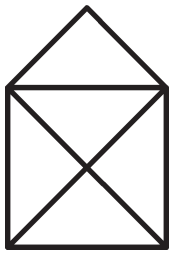
```
draw z1--z3;
draw z3--z5;
draw z5--z7;
draw z7--z1;
endchar;
end
```



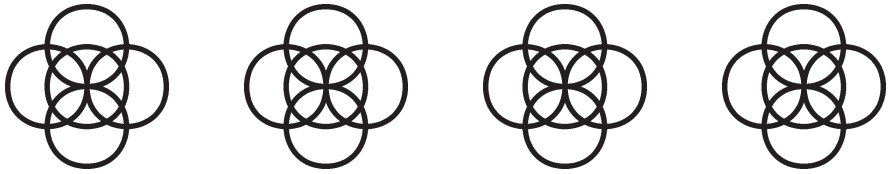
```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,10u#,10u#,0);
z1=(0,0); z2=(10u,0);z3=(10u,10u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z1;
endchar;
beginchar(2,10u#,10u#,0);
z1=(0,0); z2=(10u,0);z3=(0,10u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z1;
endchar;
beginchar(3,10u#,10u#,0);
z1=(0,0); z2=(10u,10u);z3=(0,10u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z1;
endchar;
beginchar(4,10u#,10u#,0);
z1=(10u,0); z2=(10u,10u);z3=(0,10u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z1;
endchar;
end
```



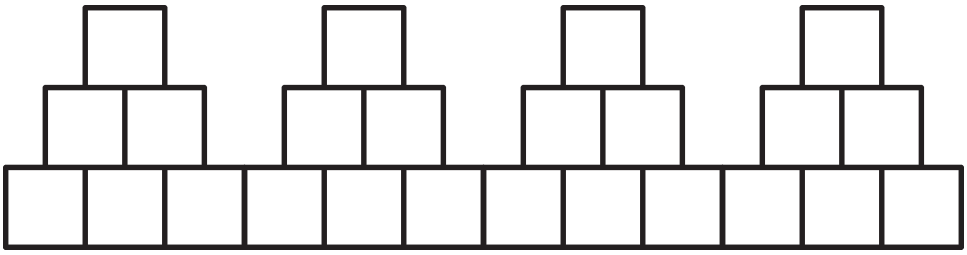
```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,10u#,10u#,0);
z1=(10u,10u); z2=(20u,10u);
pickup pencircle scaled 2pt;
draw z1..z2..z1;
endchar;
beginchar(2,10u#,10u#,0);
z1=(10u,20u); z2=(20u,20u);
pickup pencircle scaled 2pt;
draw z1..z2..z1;
endchar;
end
```



```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(20u,0);
z3=(0,20u); z4=(20u,20u);
z5=(10u,30u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z4--z5--z3--z1--z4--z2;
endchar;
end
```

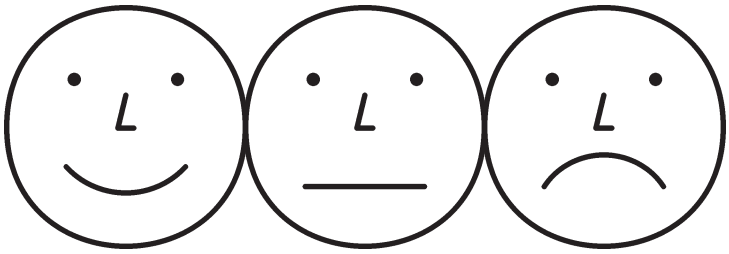


```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,10u); z2=(10u,10u);
z3=(5u,10u); z4=(15u,10u);
z5=(10u,10u); z6=(20u,10u);
z7=(10u,10u); z8=(10u,20u);
z9=(10u,0); z10=(10u,10u);
pickup pencircle scaled 2pt;
draw z1..z2..z1;
pickup pencircle scaled 2pt;
draw z3..z4..z3;
pickup pencircle scaled 2pt;
draw z5..z6..z5;
pickup pencircle scaled 2pt;
draw z7..z8..z7;
pickup pencircle scaled 2pt;
draw z9..z10..z9;
endchar;
end
```



```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(10u,0);
z3=(10u,10u); z4=(0,10u);
z5=(10u,0); z6=(20u,0);
z7=(20u,10u); z8=(10u,10u);
z9=(20u,0); z10=(30u,0);
z11=(30u,10u); z12=(20u,10u);
z13=(5u,10u); z14=(15u,10u);
z15=(15u,20u); z16=(5u,20u);
z17=(15u,10u); z18=(25u,10u);
z19=(25u,20u); z20=(15u,20u);
z21=(10u,20u); z22=(20u,20u);
z23=(20u,30u); z24=(10u,30u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z4--z1;
pickup pencircle scaled 2pt;
draw z5--z6--z7--z8--z5;
pickup pencircle scaled 2pt;
draw z9--z10--z11--z12--z9;
pickup pencircle scaled 2pt;
draw z13--z14--z15--z16--z13;
pickup pencircle scaled 2pt;
draw z17--z18--z19--z20--z17;
```

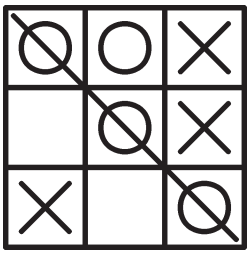
```
pickup pencircle scaled 2pt;
draw z21--z22--z23--z24--z21;
endchar;
end
```



```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(30u,0);
z4=(8u,6u); z5=(9u,6u);
z6=(21u,6u); z7=(22u,6u);
z10=(15u,4u); z11=(14u,-4);
z12=(16u,-4);
z8=(7.5u,-7.5u); z9=(22.5u,-7.5u);
pickup pencircle scaled 2pt;
draw z1..z2..z1;
pickup pencircle scaled 2pt;
draw z1..z2..z1;
pickup pencircle scaled 2pt;
draw z4..z5..z4;
pickup pencircle scaled 2pt;
```

```
fill z4..z5..cycle;
pickup pencircle scaled 2pt;
draw z6..z7..z6;
pickup pencircle scaled 2pt;
fill z6..z7..cycle;
pickup pencircle scaled 2pt;
draw z8--z9;
pickup pencircle scaled 2pt;
draw z10--z11--z12;
endchar;
beginchar(2,30u#,30u#,0);
z1=(0,0); z2=(30u,0);
z4=(8u,6u); z5=(9u,6u);
z6=(21u,6u); z7=(22u,6u);
z10=(15u,4u); z11=(14u,-4);
z12=(16u,-4);
z8=(7.5u,-5u); z9=(22.5u,-5u);
z13=(10u,-7u); z14=(20u,-7u);
pickup pencircle scaled 2pt;
draw z1..z2..z1;
pickup pencircle scaled 2pt;
draw z1..z2..z1;
pickup pencircle scaled 2pt;
draw z4..z5..z4;
pickup pencircle scaled 2pt;
fill z4..z5..cycle;
pickup pencircle scaled 2pt;
draw z6..z7..z6;
pickup pencircle scaled 2pt;
fill z6..z7..cycle;
pickup pencircle scaled 2pt;
draw z8..z13..z14..z9;
pickup pencircle scaled 2pt;
```

```
draw z10--z11--z12;
endchar;
beginchar(3,30u#,30u#,0);
z1=(0,0); z2=(30u,0);
z4=(8u,6u); z5=(9u,6u);
z6=(21u,6u); z7=(22u,6u);
z10=(15u,4u); z11=(14u,-4);
z12=(16u,-4);
z8=(7.5u,-7.5u); z9=(22.5u,-7.5u);
z13=(10u,-5u); z14=(20u,-5u);
pickup pencircle scaled 2pt;
draw z1..z2..z1;
pickup pencircle scaled 2pt;
draw z1..z2..z1;
pickup pencircle scaled 2pt;
draw z4..z5..z4;
pickup pencircle scaled 2pt;
fill z4..z5..cycle;
pickup pencircle scaled 2pt;
draw z6..z7..z6;
pickup pencircle scaled 2pt;
fill z6..z7..cycle;
pickup pencircle scaled 2pt;
draw z8..z13..z14..z9;
pickup pencircle scaled 2pt;
draw z10--z11--z12;
endchar;
end
```



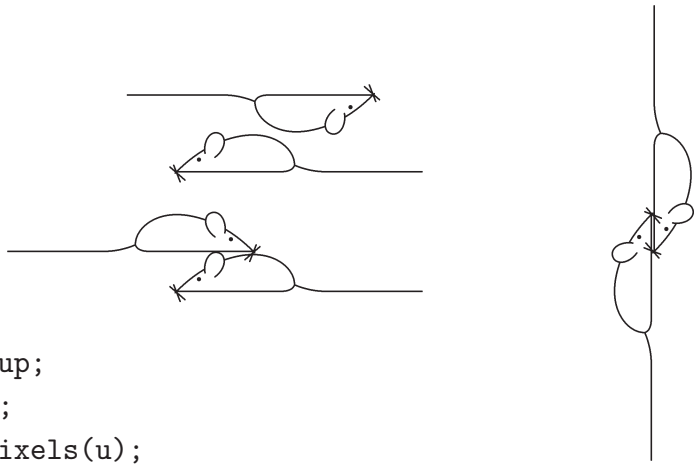
```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(30u,0);
z3=(30u,30u); z4=(0,30u);
z5=(10u,0); z6=(10u,30u);
z7=(20u,0); z8=(20u,30u);
z9=(0,10u); z10=(30u,10u);
z11=(0,20u); z12=(30u,20u);
z13=(2u,25u); z14=(8u,25u);
z15=(12u,15u); z16=(18u,15u);
z17=(22u,12u); z18=(28u,18u);
z19=(22u,18u); z20=(28u,12u);
z21=(12u,25u); z22=(18u,25u);
z23=(22u,5u); z24=(28u,5u);
z25=(2u,2u); z26=(8u,8u);
z27=(2u,8u); z28=(8u,2u);
z29=(22u,22u); z30=(28u,28u);
z31=(28u,22u); z32=(22u,28u);
z33=(1u,29u); z34=(29u,1u);
pickup pencircle scaled 2pt;
draw z1--z2--z3--z4--z1;
pickup pencircle scaled 2pt;
draw z5--z6;
pickup pencircle scaled 2pt;
```

```
draw z7--z8;
pickup pencircle scaled 2pt;
draw z9--z10;
pickup pencircle scaled 2pt;
draw z11--z12;
pickup pencircle scaled 2pt;
draw z13..z14..z13;
pickup pencircle scaled 2pt;
draw z15..z16..z15;
pickup pencircle scaled 2pt;
draw z17--z18;
pickup pencircle scaled 2pt;
draw z19--z20;
pickup pencircle scaled 2pt;
draw z21..z22..z21;
pickup pencircle scaled 2pt;
draw z23..z24..z23;
pickup pencircle scaled 2pt;
draw z25--z26;
pickup pencircle scaled 2pt;
draw z27--z28;
pickup pencircle scaled 2pt;
draw z29--z30;
pickup pencircle scaled 2pt;
draw z31--z32;
pickup pencircle scaled 2pt;
draw z33--z34;
endchar;
end
```

2.5. Proměnná picture

Námět myši byl převzat z [2]. V následující části vysvětlím, jak pracovat s proměnnou typu `picture`. V prvním kroku vykreslím myš a poté ji uložím jako hodnotu proměnné `picture`. Tuto proměnnou musíme deklarovat (viz. níže). Proměnná `picture` se používá, pokud chceme obrázek různě transformovat (např. otáčet, posouvat, roztahovat nebo zmenšovat).

```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(1,30u#,10u#,0);
picture mouse;
pickup pencircle scaled .2u;
draw ((0,.4)---(25,.4)..{dir70}(30,2){up}..(26,8)..(15,9)
..{dir226}origin)scaled .5u;
draw ((30,2)..(38,.4)---(62,.4))scaled .5u;
erase fill ((7.5,5){dir100}..(11,10)..{dir215}(10,4)--cycle)
scaled .5u;
draw ((7.5,5){dir100}..(11,10)..{dir215}(10,4))scaled .5u;
draw (3dir-35--2dir130)scaled .5u;
draw (2.5dir-60--2dir100)scaled .5u;
fill fullcircle scaled .5u shifted (3u,1.7u);
endchar;
end
```



```
mode_setup;
u#:=3pt#;
define_pixels(u);
screenstrokes;
input osy;
beginchar(2,60u#,40u#,0);
picture mouse;
pickup pencircle scaled .2u;
draw ((0,.4)---(25,.4)..{dir70}(30,2){up}..(26,8)..(15,9)
..{dir226}origin)scaled .5u;
draw ((30,2)..(38,.4)---(62,.4))scaled .5u;
erase fill ((7.5,5){dir100}..(11,10)..{dir215}(10,4)--cycle)
scaled .5u;
draw ((7.5,5){dir100}..(11,10)..{dir215}(10,4))scaled .5u;
draw (3dir-35--2dir130)scaled .5u;
draw (2.5dir-60--2dir100)scaled .5u;
fill fullcircle scaled .5u shifted (3u,1.7u);
mouse=currentpicture;
addto currentpicture also mouse shifted(0,15u);
addto currentpicture also mouse rotated 180 shifted (25u,25u);
addto currentpicture also mouse
reflectedabout((5u,0),(5u,20u))shifted(5u*up);
addto currentpicture also mouse
reflectedabout(origin,dir45)shifted(w,5u);
addto currentpicture also mouse
reflectedabout(origin,dir-45)shifted(w,10u);
endchar;
end
```

3. Kombinatorické struktury

Kombinatorické struktury jsou struktury sestavené opakováním konečného počtu elementů. Nejdříve tedy vytvoříme elementy. V našem případě trojúhelníkové, půlkruhové a lichoběžníkové. Dále jednotlivé elementy seskládáme v různém pořadí, vznikne námi zmiňovaná kombinatorická struktura.

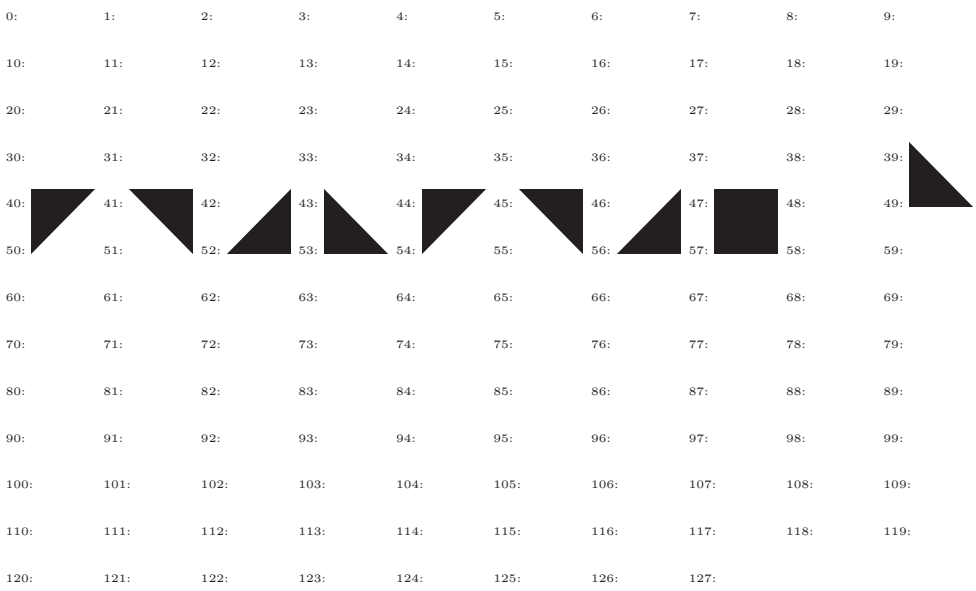
3.1. Sébastien Truchet

Sébastien Truchet použil r. 1704 diagonálně rozdělené čtvercové elementy doplněné prázdným a plným (např. bílým a černým) elementem (tedy celkem šest různých typů prvků) k sestavení „kombinatorických“ struktur.

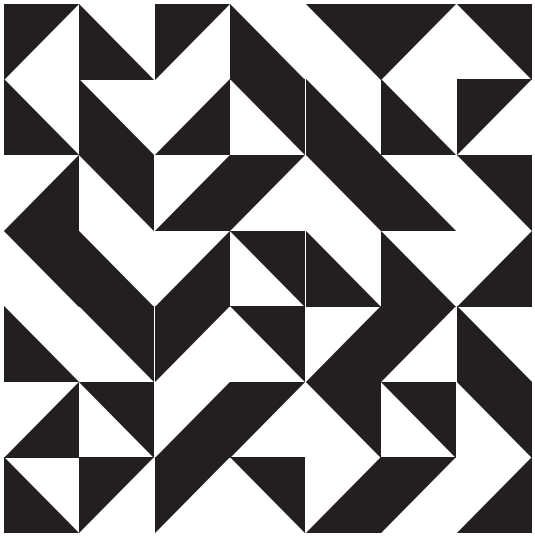
Můžeme napodobit jeho práci. V našem fontu navíc přidáme 4 trojúhelníkové elementy, abychom pokryli všech 9 číslic, které použijeme k sestavení obrázku.

```
mode_setup;
u#:=1mm#;
define_pixels(u);
input osy.mf ;
path p[];
z1=(0,0); z2=(0,10u); z3=(10u,10u); z4=(10u,0);
p1:=z1--z2--z4--cycle;
p2:=z1--z2--z3--z4--cycle;
pickup pencircle scaled d0;
beginchar("0",10u#,10u#,0);  %prázdný
endchar;
beginchar("1",10u#,10u#,0);
filldraw p1;  %draw p2;
endchar;
beginchar("2",10u#,10u#,0);
filldraw p1;  %draw p2;
currentpicture:=currentpicture rotatedaround((5u,5u), -90);
endchar;
beginchar("3",10u#,10u#,0);
filldraw p1;  %draw p2;
currentpicture:=currentpicture rotatedaround((5u,5u), 180);
endchar;
beginchar("4",10u#,10u#,0);
filldraw p1;  %draw p2;
```

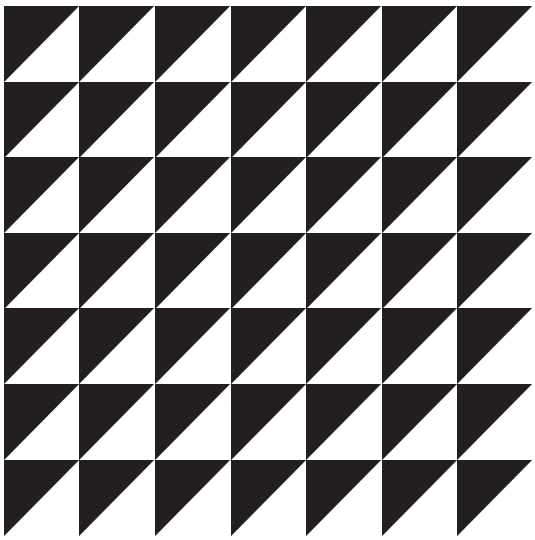
```
currentpicture:=currentpicture rotatedaround((5u,5u), 90);
endchar;
beginchar("5",10u#,10u#,0);  %opakujeme
filldraw p1;  %draw p2;
endchar;
beginchar("6",10u#,10u#,0);
filldraw p1;  %draw p2;
currentpicture:=currentpicture rotatedaround((5u,5u), -90);
endchar;
beginchar("7",10u#,10u#,0);
filldraw p1;  %draw p2;
currentpicture:=currentpicture rotatedaround((5u,5u), 180);
endchar;
beginchar("8",10u#,10u#,0);
filldraw p1;  %draw p2;
currentpicture:=currentpicture rotatedaround((5u,5u), 90);
endchar;
beginchar("9",10u#,10u#,0);  %plný
filldraw  p2;
endchar;
end;
```

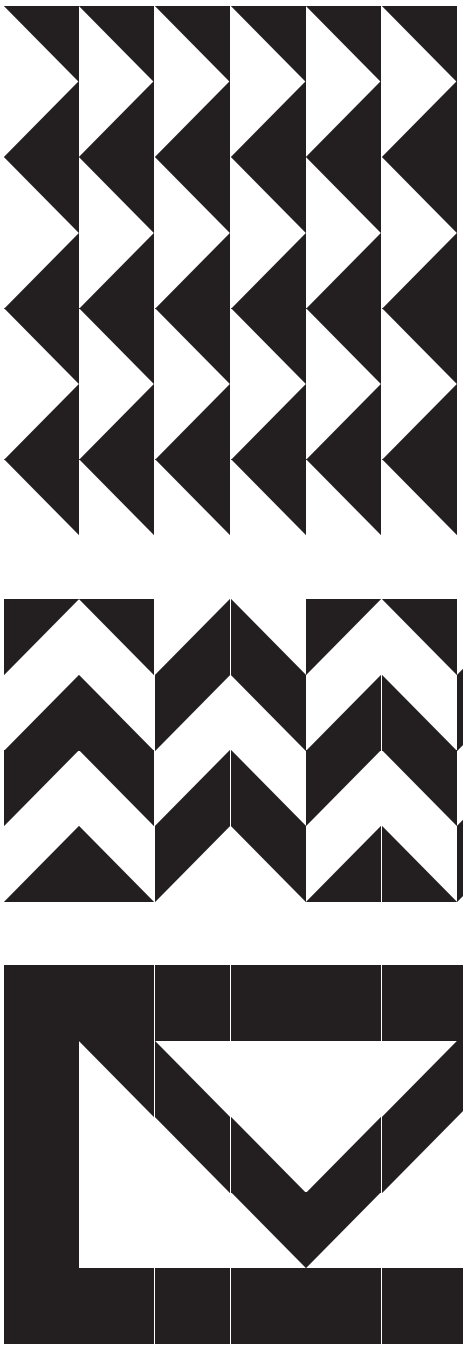


```
{\elementy\offinterlineskip
\hbox{6121323}
\hbox{1143512}
\hbox{4342313}
\hbox{3143514}
\hbox{1323425}
\hbox{4342333}
\hbox{5223424}
}}
```



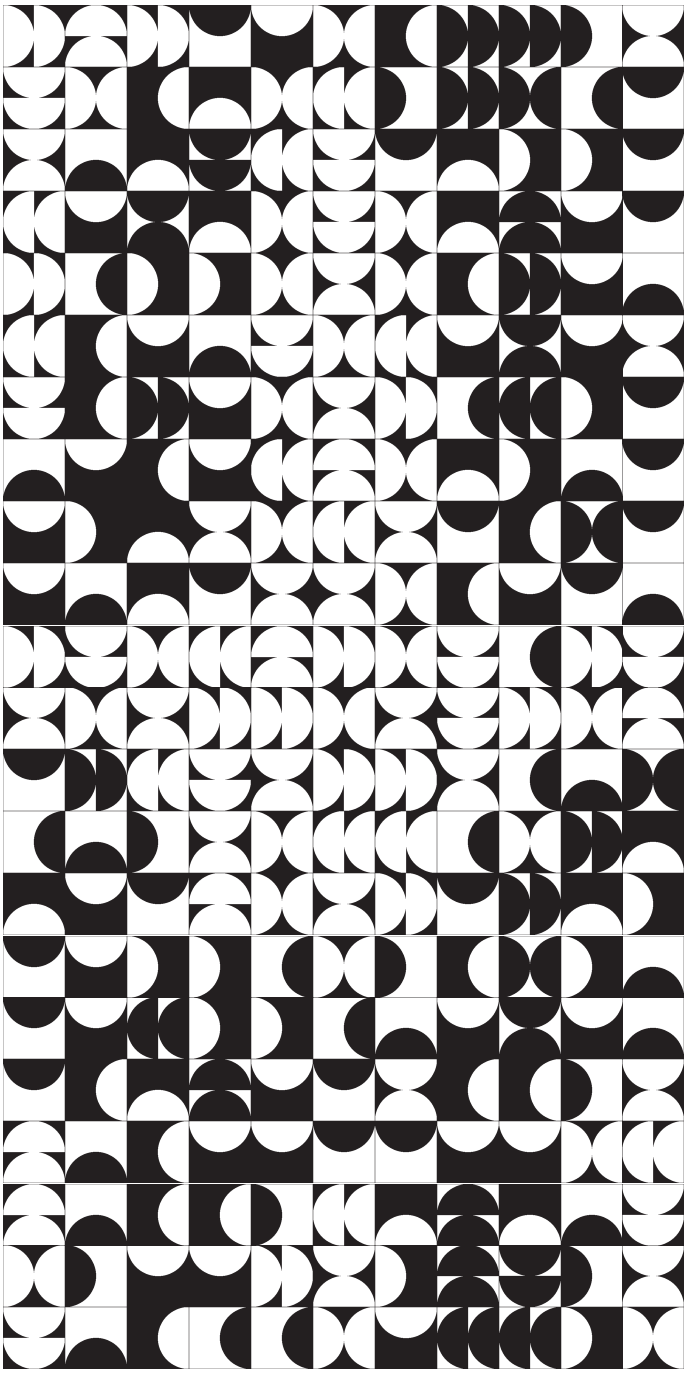
Jiné Truchetem inspirované struktury:

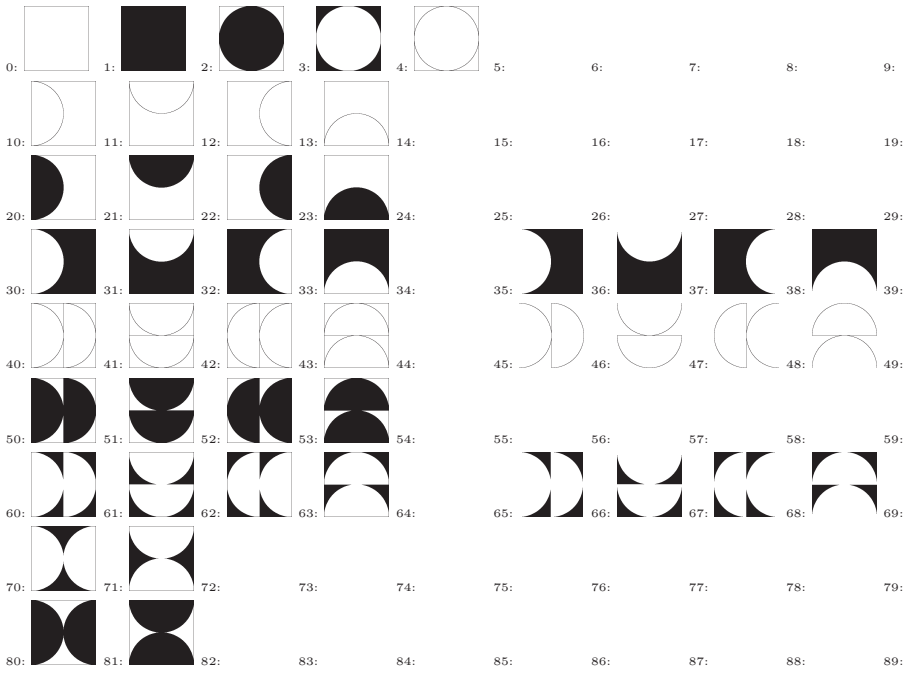




3.2. Zdeněk Sýkora

Kombinatorické struktury v jedné fázi své tvorby vytvářel i malíř Zdeněk Sýkora. Pomocí obrázkového fontu můžeme snadno zopakovat jeho černo-bílou strukturu:

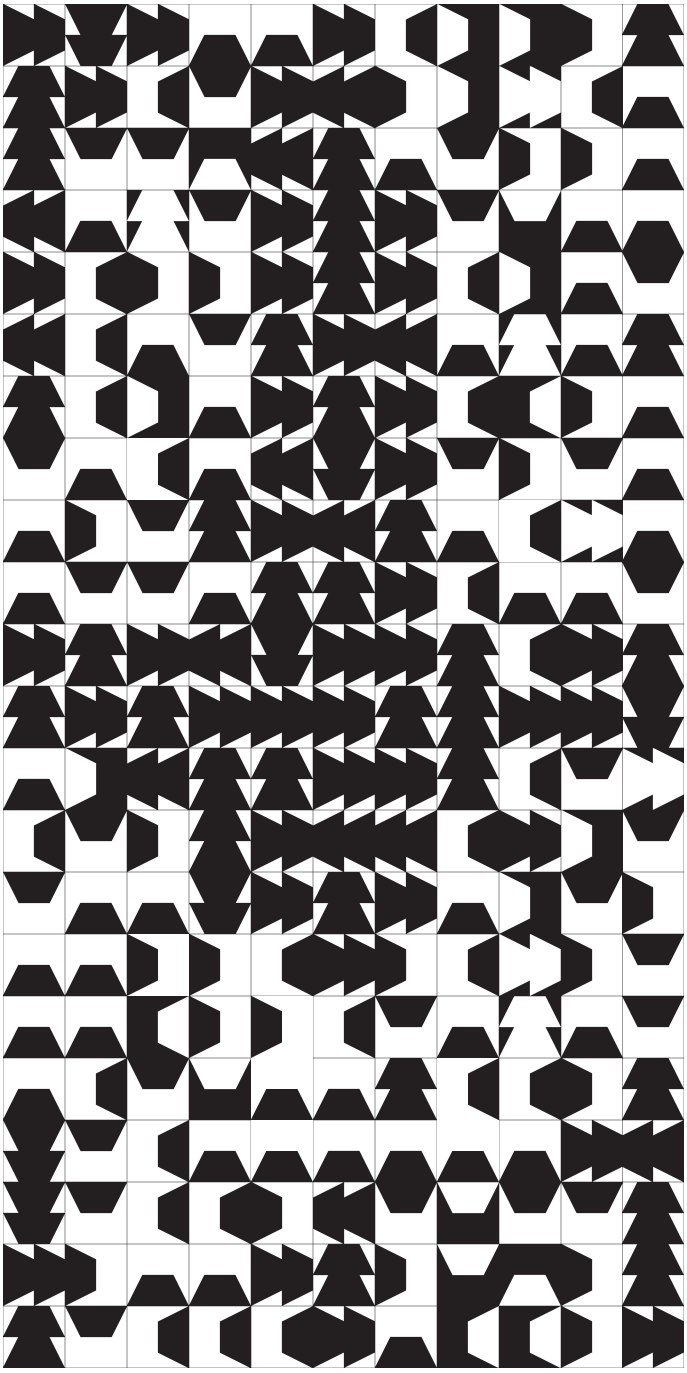


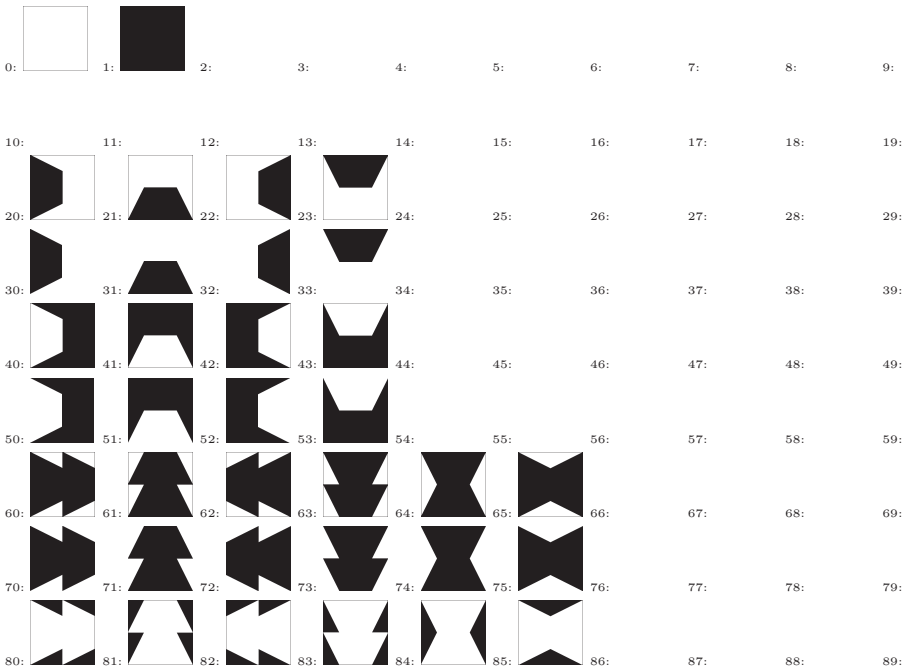


Místo trouchetovských elementů jsme vytvořili elementy půlkruhové. Ukázka kódu jednoho znaku:

```
%%%%%%%%%%%%%%%%%%%%%%%%%% plne polokruhy
beginchar(20,10u#,10u#,0);
path p[];
z1=(0,0); z2=(0,10u); z3=(10u,10u); z4=(10u,0);
p1:=z1--z2--z3--z4--cycle;
p2:=halfcircle scaled 10u shifted (5u,0)--cycle;
pickup pencircle scaled d0;
draw p1;
filldraw p2;
currentpicture:=currentpicture rotatedaround((5u,5u), 270);
endchar;
```

Můžeme připravit i lichoběžníkové elementy a Sýkorovu strukturu modifikovat.





Ukázka kódu čtyř znaků lišících se pouze o pootočení:

```
%%%%%% prazdnelichobezniky, plny zbytek
beginchar(40,10u#,10u#,0);
path p[];
z1=(0,0); z2=(0,10u); z3=(10u,10u); z4=(10u,0);
z5=(2.5u,5u); z6=(7.5u,5u);
p1:=z1--z2--z3--z4--cycle;
p2:=z1--z4--z6--z5--cycle;
pickup pencircle scaled d0;
filldraw p1;
unfilldraw p2;
draw p1;  %%%% rám
currentpicture:=currentpicture rotatedaround((5u,5u), 270);
endchar;

beginchar(41,10u#,10u#,0);
path p[];
z1=(0,0); z2=(0,10u); z3=(10u,10u); z4=(10u,0);
z5=(2.5u,5u); z6=(7.5u,5u);
```

```
p1:=z1--z2--z3--z4--cycle;
p2:=z1--z4--z6--z5--cycle;
pickup pencircle scaled d0;
filldraw p1;
unfilldraw p2;
draw p1;
endchar;

beginchar(42,10u#,10u#,0);
path p[];
z1=(0,0); z2=(0,10u); z3=(10u,10u); z4=(10u,0);
z5=(2.5u,5u); z6=(7.5u,5u);
p1:=z1--z2--z3--z4--cycle;
p2:=z1--z4--z6--z5--cycle;
pickup pencircle scaled d0;
filldraw p1;
unfilldraw p2;
draw p1;
currentpicture:=currentpicture rotatedaround((5u,5u), 90);
endchar;

beginchar(43,10u#,10u#,0);
path p[];
z1=(0,0); z2=(0,10u); z3=(10u,10u); z4=(10u,0);
z5=(2.5u,5u); z6=(7.5u,5u);
p1:=z1--z2--z3--z4--cycle;
p2:=z1--z4--z6--z5--cycle;
pickup pencircle scaled d0;
filldraw p1;
unfilldraw p2;
draw p1;
currentpicture:=currentpicture rotatedaround((5u,5u), 180);
endchar;
```

Závěr

Ve své práci jsem se snažila přiblížit čtenářům program METAFONT, který není tak známý, ale je velice užitečný k tvorbě čárové grafiky. V úvodu jsem vysvětlila, jak tento program spolupracuje s TEXem, také jsem se zmínila o otci těchto programů Donaldu Ervinu Knuthovi. I díky jeho knize [1] tato práce mohla vzniknout.

V práci začínám prostým kreslením čar nebo šipek. Dále se úroveň zvyšuje, kreslím geometrické tvary. Těmto geometrickým tvarům jsem věnovala hodně prostoru, ne z toho důvodu, že by vykreslení bylo příliš složité. Chtěla jsem nejen ukázat, jak s METAFONTem naložit tak, aby měly obrázky správnou velikost, nebo aby vypadaly jako v některé z příruček, ale vytvářela jsem je jako přípravu pro programování elementů kombinatorických struktur. Nakonec jsem dala prostor i své fantazii (té se zde meze nekladou) a vykreslila několik – doufám, že povedených – obrázků. Inspiraci jsem hledala hlavně ve věcech, které mě obklopují (i když z obrázků to není příliš patrné). V neposlední řadě jsem si vyzkoušela práci s proměnnou typu `picture`. To bylo pro mě něco nového, ale po předešlých zkušenostech se mi práce dařila. V závěru práce se pokouším napodobit dílo malíře Zdeňka Sýkory, který vytvářel kombinatorické struktury inspirované Sébastienem Truchetem (i jemu se v práci věnuji).

V době, kdy jsem se seznamovala s obsahem své budoucí práce, jsem toho o kombinatorických strukturách mnoho nevěděla, byly pro mě z říše „science fiction“. Nyní takové struktury dokáži vytvářet sama.

Musím přiznat, že tato bakalářská práce pro mě nebyla pouze povinností, jak se mi zprvu zdálo, ale vyklubala se z ní pro mě skvělá zábava. A co víc, ve věcech kolem sebe vidím i METAFONTový zdrojový text.

Literatura

[1] D. Knuth: The MetafontBook. Addison-Wesley, 1992.

[2] P. Šedivý a kol.: Kreslíme Metafontem. Zpravodaj CSTUG, 1 (1998).

[3] http://en.wikipedia.org/wiki/Donald_Knuth