FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Datové struktury

Zuzana Majdišová

19.5.2015

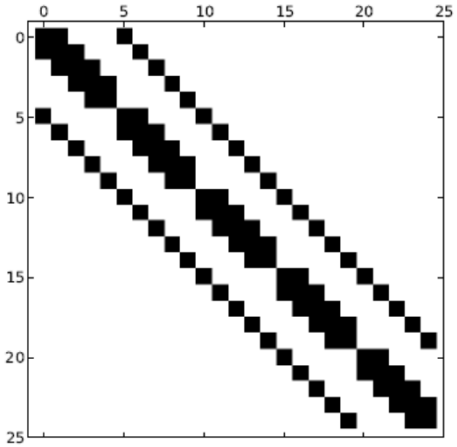
Datové struktury

- Numerické datové struktury
  - Efektivní reprezentace velkých řídkých matic
  - Lze využít při výpočtu na GPU
- Dělení prostoru a binární masky
- Voxelová datová struktura

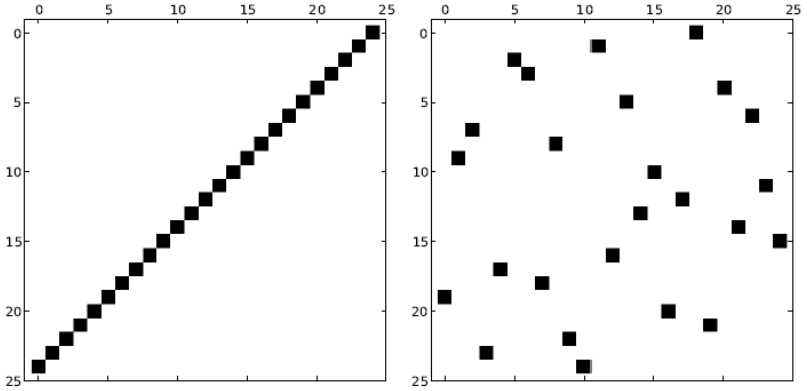
DIA – Diagonální formát

- Definován poli: `data` (nenulové prvky) a `offsets`

Offset:	0	i > 0	i < 0
	Hlavní diagonála	super-diagonála	sub-diagonála



- **Výhody:** `row` a `col` definovány implicitně
  souvislý přístup k paměti
- **Nevýhody:** vhodné pouze pro určité uspořádání

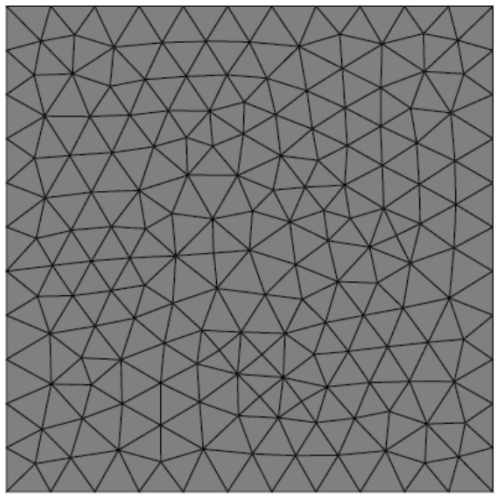


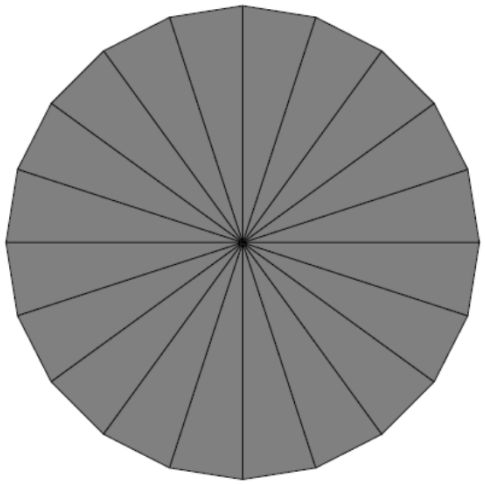
$$\texttt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix} \quad \texttt{data} = \begin{bmatrix} * & 1 & 7 \\ * & 2 & 8 \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \quad \texttt{offsets} = \begin{bmatrix} -2 & 0 & 1 \end{bmatrix}$$

$$\begin{array}{rl} \texttt{data} & \begin{bmatrix} * & * & 5 & 6 & 1 & 2 & 3 & 4 & 7 & 8 & 9 & * \end{bmatrix} \\ \text{Iteration 0} & \begin{bmatrix} & & 2 & 3 & & & & & & & & \end{bmatrix} \\ \text{Iteration 1} & \begin{bmatrix} & & & & 0 & 1 & 2 & 3 & & & & \end{bmatrix} \\ \text{Iteration 2} & \begin{bmatrix} & & & & & & & & 0 & 1 & 2 & \end{bmatrix} \end{array}$$

ELL – ELLPACK formát

- Vhodný pro $M \times N$ matici s max. K nenulovými prvky v řádku
  $\Rightarrow$ Definován poli: `data` ($M \times K$) a `indices` (sloupcové indexy)
- **Výhody:** obecnější než DIA
- **Nevýhody:** potřeba přibližně stejný počet nenulových prvků v každém řádku
- **Využití**: matice definovaná stencil operací pro síť s téměř stejnou konektivitou vrcholů





$$\texttt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix} \quad \texttt{data} = \begin{bmatrix} 1 & 7 & * \\ 2 & 8 & * \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \quad \texttt{indices} = \begin{bmatrix} 0 & 1 & * \\ 1 & 2 & * \\ 0 & 2 & 3 \\ 1 & 3 & * \end{bmatrix}$$

$$\begin{array}{rl} \texttt{data} & [1\ 2\ 5\ 6\ 7\ 8\ 3\ 4\ *\ *\ 9\ *] \\ \texttt{indices} & [0\ 1\ 0\ 1\ 1\ 2\ 2\ 3\ *\ *\ 3\ *] \\ \text{Iteration 0} & [0\ 1\ 2\ 3\ \quad\quad\quad\quad\quad\quad\quad\quad] \\ \text{Iteration 1} & [\quad\quad\quad\quad 0\ 1\ 2\ 3\ \quad\quad\quad\quad] \\ \text{Iteration 2} & [\quad\quad\quad\quad\quad\quad\quad\quad 0\ 1\ 2\ 3] \end{array}$$

COO – Souřadnicový (XY) formát

- Jednoduché schéma
- Definován poli: `data` (nenulové prvky), `row` (řádkové indexy) a `col` (sloupcové indexy)
- **Výhody:** lze reprezentovat libovolnou řídkou matici
- **Nevýhody:** `row` a `col` definovány explicitně $\Rightarrow$ vyšší výpočetní náročnost

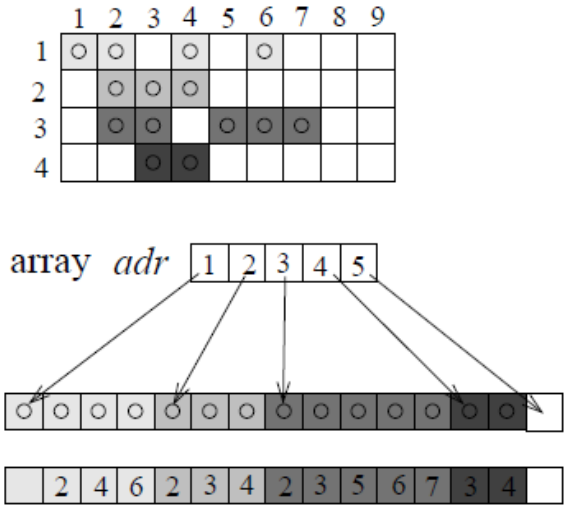
$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\mathtt{row} = [0\ 0\ 1\ 1\ 2\ 2\ 2\ 3\ 3]$$
$$\mathtt{col} = [0\ 1\ 1\ 2\ 0\ 2\ 3\ 1\ 3]$$
$$\mathtt{data} = [1\ 7\ 2\ 8\ 5\ 3\ 9\ 6\ 4]$$

`row`	$[0\ 0\ 1\ 1\ 2\ 2\ 2\ 3\ 3]$
`col`	$[0\ 1\ 1\ 2\ 0\ 2\ 3\ 1\ 3]$
`data`	$[1\ 7\ 2\ 8\ 5\ 3\ 9\ 6\ 4]$
Iteration 0	$[0\ 1\ 2\ 3\ \ \ \ \ \ \ \ ]$
Iteration 1	$[\ \ \ \ \ \ \ \ 0\ 1\ 2\ 3\ ]$
Iteration 2	$[\ \ \ \ \ \ \ \ \ \ \ \ \ \ 0]$

CSR – Compressed Sparse Row formát

- „rozšíření“ COO formátu – aplikace komprese
- Definován poli: `data` (nenulové prvky), `ptr` (pointery) a `indices` (sloupcové indexy)
- **Výhody:** rychlejší zjišťování hodnot matice
  lze jednoduše zjistit počet nenulových prvků v řádku…



$$A = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\texttt{ptr} = [0\ \ 2\ \ 4\ \ 7\ \ 9]$$

$$\texttt{indices} = [0\ \ 1\ \ 1\ \ 2\ \ 0\ \ 2\ \ 3\ \ 1\ \ 3]$$

$$\texttt{data} = [1\ \ 7\ \ 2\ \ 8\ \ 5\ \ 3\ \ 9\ \ 6\ \ 4]$$

indices [0 1 1 2 0 2 3 1 3]

data [1 7 2 8 5 3 9 6 4]

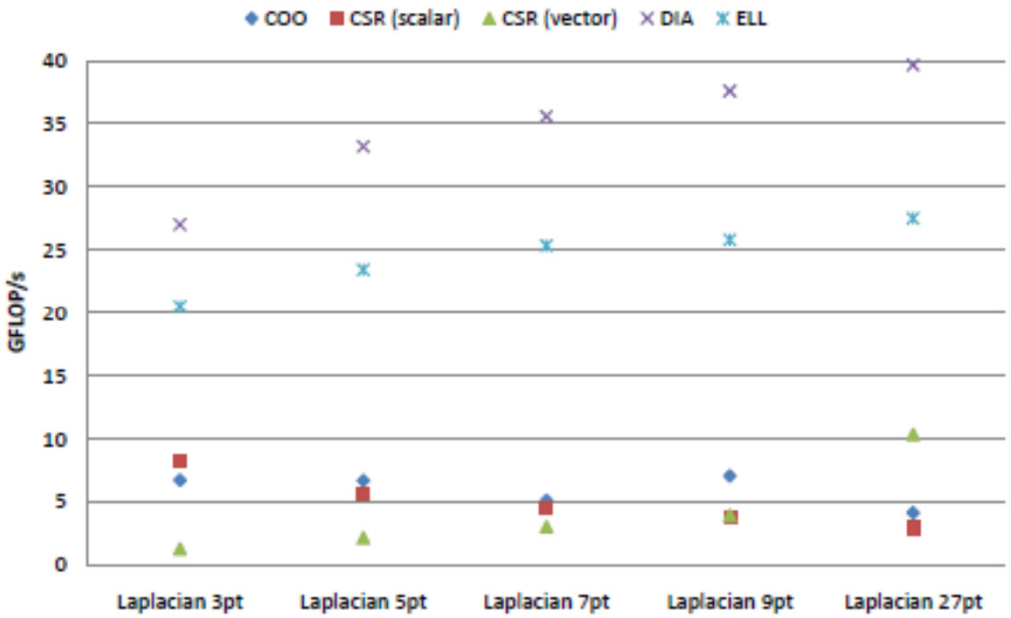
Iteration 0	[0 1 2 3]	Warp 0	[0 0]
Iteration 1	[0 1 2 3]	Warp 1	[1 1]
Iteration 2	[2]	Warp 2	[2 2 2]
		Warp 3	[3 3]

HYB – Hybridní formát

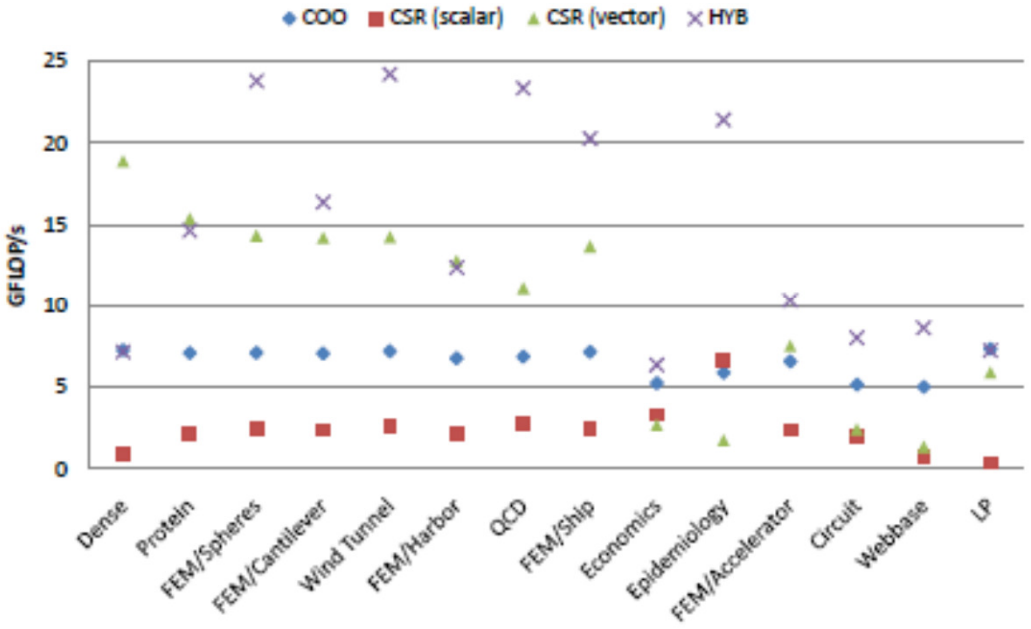
- Kombinace ELL a COO formátu
- Většina nenulových prvků uložena do ELL formátu, zbytek do COO formátu
- Nejlepší K pro ELL formát určeno z histogramu
- Plně obsazený ELL formát je přibližně 3 krát rychlejší než COO formát
  $\Rightarrow K$ sloupců pro ELL formát pokud má alespoň třetina řádků víc než K nenulových prvků

Experimenty – matice ×vektor

Strukturované matice



Nestrukturované matice



Quadtree storage format

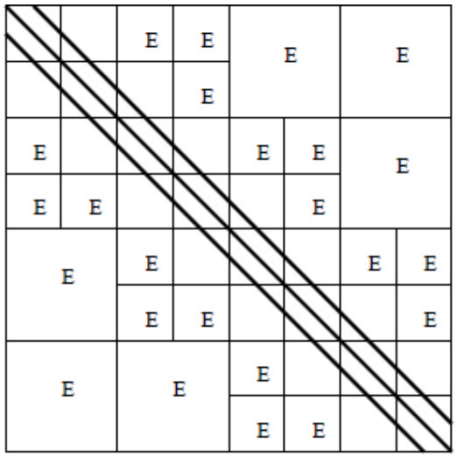
- Předchozí struktury pomalé důsledkem nepřímé adresace
- Řídké matice často obsahují husté podmatice (bloky) ⇒ vznik blokových formátů
- Využití quadtree – uzly odpovídají podmaticím
- **Typu uzlů:** vnitřní:
  - „Mixed“
  - „Empty“ – NULL pointer

  listy:
  - „Full“ – podíl nenulových prvků větší než práh
  - „Sparse“ – podmatice reprezentována COO formátem
- **Výhody:** alokováno méně paměti

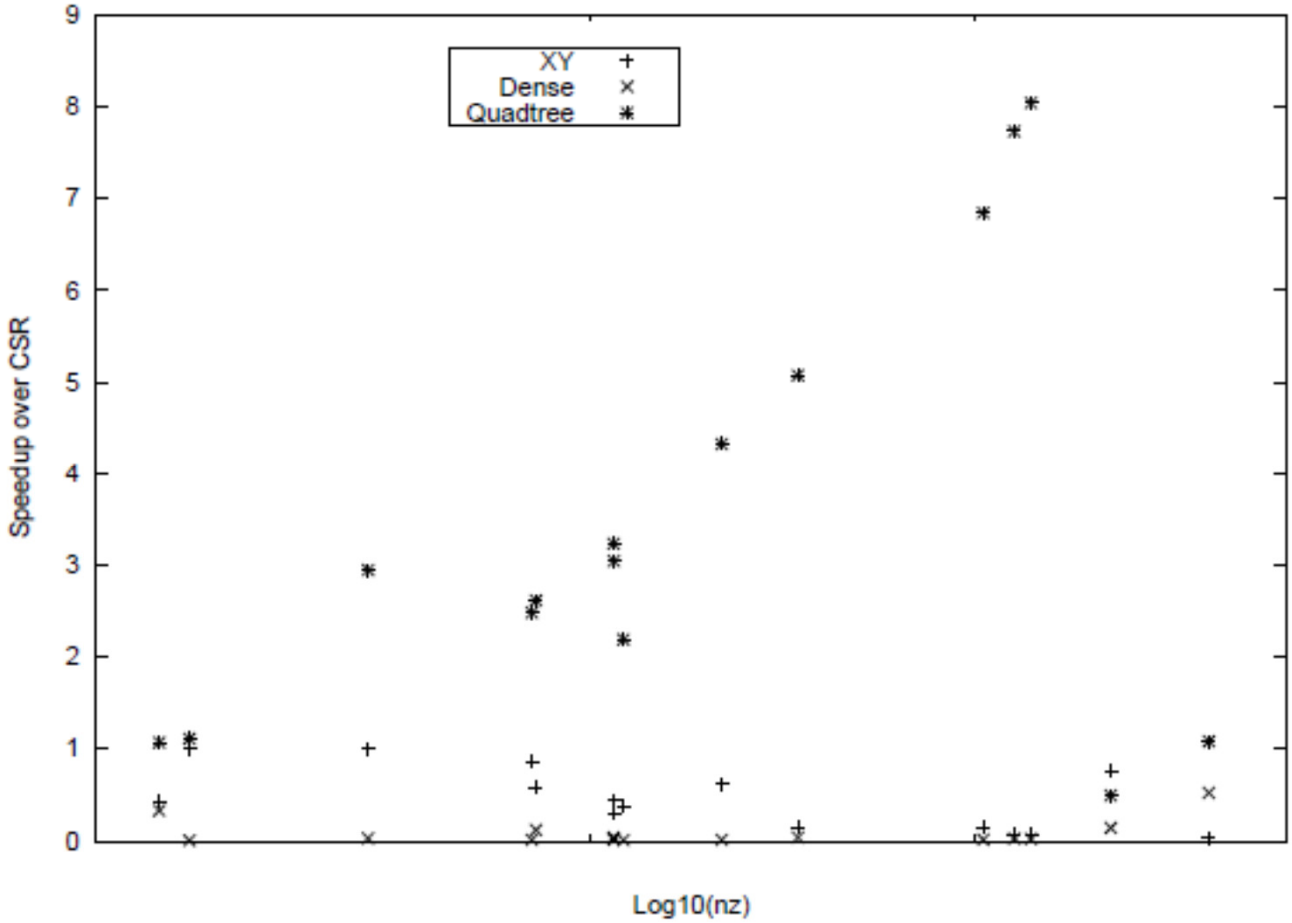
  matice uložena jako malé husté bloky ⇒ lepší výkon

  jednoduchá konverze z předchozích formátů

  relativně jednoduchá modifikace quadtree

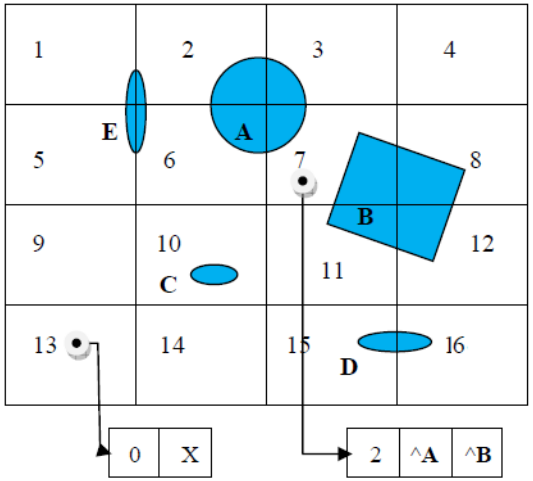


Experimenty - matice × matice



Dělení prostoru

- Jednoduchá technika dělící (rovnoměrně) prostor na menší prostory
- Paměťová náročnost: $O\left(p\ q\ M^d\right)$
  - p – počet objektů
  - d – dimenze
  - M – počet dělení v jedné ose
  - $q = q(M, p, s)$ – pravděpodobnost, že „průměrný“ objekt protne podprostor
- Paměťové nároky prudce rostou s jemností dělení a dimenzí prostoru



Rezidenční maska

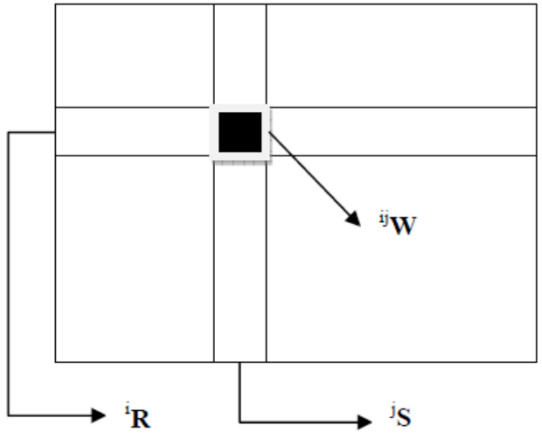
- Pro každý objekt definován vektor bitů
- Délka vektoru bitů dána celkovým dělením prostoru
- Bit nastaven na “1“ pokud objekt protíná daný interval
- Paměťová nároky: $M_{RM} = pM^d/8\ \ [B]$
  - p – počet objektů
  - d – dimenze
  - M – počet dělení v jedné ose

$$\boldsymbol{Q} = \begin{bmatrix} 0000 & 0000 & 0110 & 0110 \\ 0000 & 1100 & 1100 & 0000 \\ 0000 & 0010 & 0000 & 0000 \\ 1100 & 0000 & 0000 & 0000 \\ 0000 & 0000 & 0011 & 0011 \\ 15 & bits & \dots\dots & 0 \end{bmatrix} = \begin{bmatrix} \text{Object } \mathbf{A} \\ \text{Object } \mathbf{B} \\ \text{Object } \mathbf{C} \\ \text{Object } \mathbf{D} \\ \text{Object } \mathbf{E} \\ \end{bmatrix}$$

- Časová složitost: $O\left(p\ M^d\right)$
- **Výhody:** rychlá detekce kolize objektů pomocí $\mathbf{land}$
- **Nevýhody:** při jemném dělení a malé velikosti objektů velmi dlouhé vektory bitů obsahující většinou nuly

Binární maska

- Nechť ${}^{i}\boldsymbol{R}$, resp. ${}^{j}\boldsymbol{S}$ – množina objektů protínajících i-tý řádkový řez resp. j-tý sloupcový řez, pak všechny objekty protínající buňku na pozici (i,j) dány: ${}^{ij}\boldsymbol{W} = {}^{i}\boldsymbol{R} \cap {}^{j}\boldsymbol{S}$
- Množiny ${}^{i}\boldsymbol{R}$, resp. ${}^{j}\boldsymbol{S}$ reprezentovány vektorem bitů (k-tý bit reprezentuje k-tý objekt) $\Rightarrow {}^{ij}\boldsymbol{W} = {}^{i}\boldsymbol{R}\ \mathbf{land}\ {}^{j}\boldsymbol{S}$
- Paměťová nároky: $M_{BM} = dpM/8\ [B]$
  - p – počet objektů
  - d – dimenze
  - M – počet dělení v jedné ose



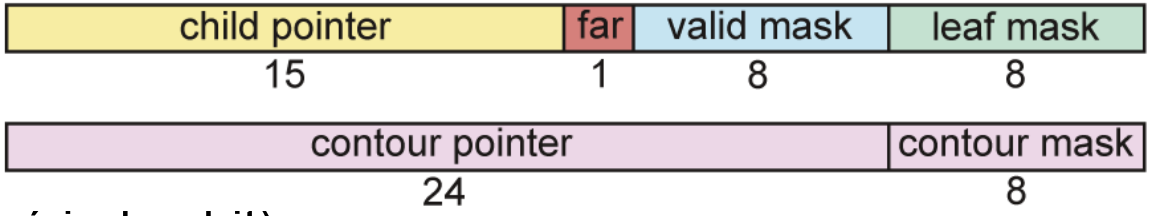
- **Výhody:** rychlé nalezení všech objektů protínajících danou oblast
  jednoduchá kontrola konzistence dat

Voxelová datová struktura

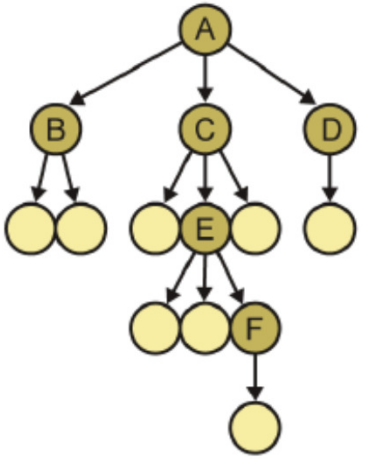
- Voxely uložíme pomocí řídkého octree (uzel reprezentuje voxel)
- Využití ray casting
- Data dělena do bloků obsahujících:
  - Pole child descriptors
    - Kódování topologie octree
  - Info sekce
    - Adresář dostupných attachments
    - Pointer na první child descriptor
  - Contour data
    - Popis geometrického tvaru voxelů
  - Různé množství attachments
    - Pole ukládající atributy pro stínování pro každý voxel

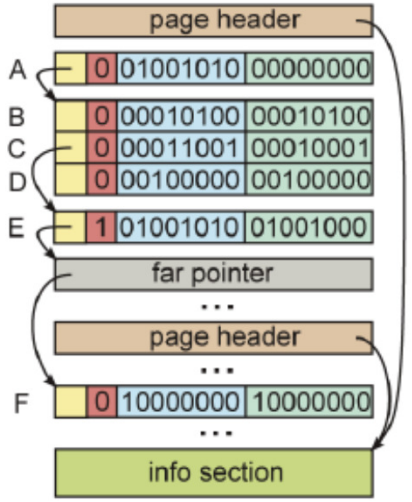
Child descriptor (64-bit)

- Každý odpovídá jednomu nelistovému voxelu (list popsán ve svém rodiči)
- Rozdělen na dvě 32-bit části:
  1. Popis množiny potomků
  2. Souvisí s contours



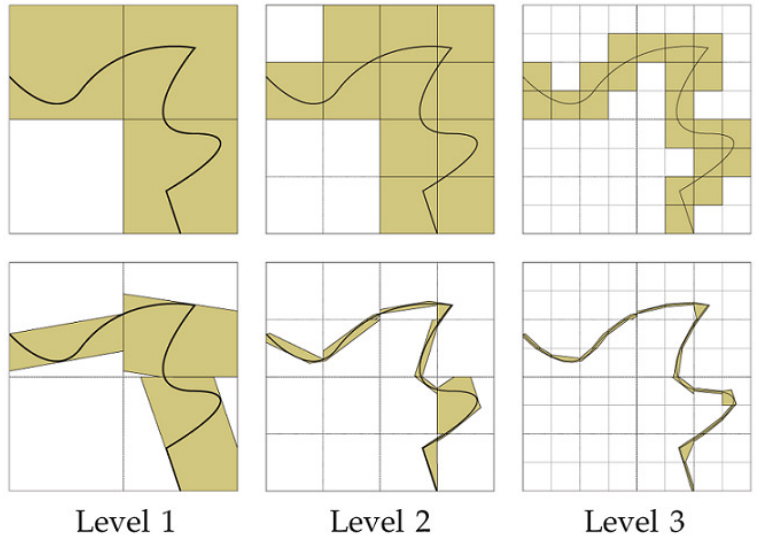
- Dvě bitové masky (každý potomek má jeden bit)
  - *valid mask* – "1" pokud potomek obsahuje voxel
  - *leaf mask* – "1" pokud potomek je list
- *Child pointer* (15-bit)
  - existuje nelistový potomek ⇒ pak pointer na child descriptor
- *Far* bit
  - pokud nastaven *child pointer* je nepřímý odkaz na far pointer

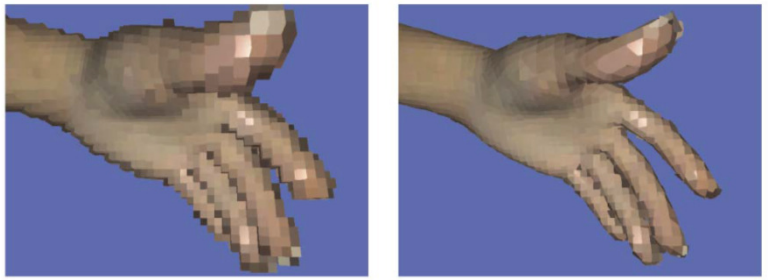




Contours

- Přímočará vizualizace voxelových dat je aproximace geometrie uvnitř voxelu krychličkou
  - Vznik aproximační chyby
- Redukce chyby:
  - Voxel oříznut dvojící rovnoběžných rovin (contour) odpovídajících orientaci aproximované plochy
  - Výsledkem kolekce orientovaných slabs
- *Contour mask*
  - "1" pokud potomek má přidruženu contour
- *Contour pointer* (24-bit)
  - Odkaz na seznam contours (pro každý nastavený bit jedna)
- Uložení contour (32-bit)
  - 3 krát 6-bit int pro definování normálových vektorů dvou rovin
  - 2 krát 7-bit int definujících jejich pozici uvnitř voxelu





FAKULTA APLIKOVANÝCH VĚD ZÁPADOČESKÉ UNIVERZITY V PLZNI

Budoucí práce

- Vývoj datových struktur pro meshfree techniky
  - Využití projektivní geometrie
  - Využití geometrické algebry

DĚKUJI ZA POZORNOST

FAKULTA APLIKOVANÝCH VĚD ZÁPADOČESKÉ UNIVERZITY V PLZNI

Reference

- Bell,N.,Garland,M.: Efficient Sparse Matrix-Vector Multiplication on CUDA, Nvidia Technical Report NVR-2008-004, Nvidia Corporation, Vol.2, No.5, 2008.
- Bell,N.,Garland,M.: Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors, Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM, p.18, 2009.
- Šimeček,I.: Sparse Matrix Computations using the Quadtree Storage Format, Proceedings of 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2009), pp.168-173, 2009.
- Skala,V.: An Efficient Space Partitioning Method Using Binary Maps, 11th Conference SIP'2012 conference , pp.121-124, ISBN: 978-1-61804-081-7, St.Malo, WSEAS, France, 2012.
- Laine,S., Tero,K.: Efficient Sparse Voxel Octrees, Visualization and Computer Graphics, IEEE Transactions on, Vol.17, No.8, pp.1048-1059, 2011.