Struktura scény

Petr Felkel
Katedra počítačové grafiky a interakce, ČVUT FEL
místnost KN:E-413 (Karlovo náměstí, budova E)
E-mail: felkel@fel.cvut.cz

S použitím materiálů Bohuslava Hudce, Jaroslava Sloupa a Vlastimila Havrana

Poslední změna: 3.5.2016

Osnova

Struktura scény

- Modelování a reprezentace scény
  - Lineární (nehierarchická) reprezentace
  - Hierarchická reprezentace
  - Vlastní graf scény – příprava pro cvičení

Modelování

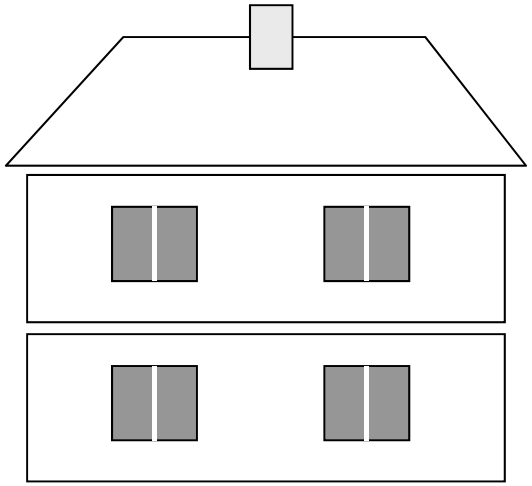
- **modely** = abstrakce virtuálních světů, které vytváříme svými programy
- v počítačové grafice modelujeme pomocí **geometrických objektů**
- většina API poskytuje naprosté minimum geometrických primitiv, složitější objekty z nich musí vytvořit uživatel
  (OpenGL pouze trojúhelník, čára a bod)
- **Složené objekty v modelech a vazby mezi nimi musíme nějak reprezentovat SAMI**
- Objekty se typicky načtou z externích souborů pomocí knihoven, například knihovna ASSIMP: http://assimp.sourceforge.net/

  Příklad použití:
  http://www.lighthouse3d.com/cg-topics/code-samples/importing-3d-models-with-assimp/
  (Pozn. Knihovna je pomalá v debug režimu na windows)

Modelování

geometrický objekt (dům)

lze rozdělit na nezávislé **segmenty** (detaily) – např. okno, komín,...

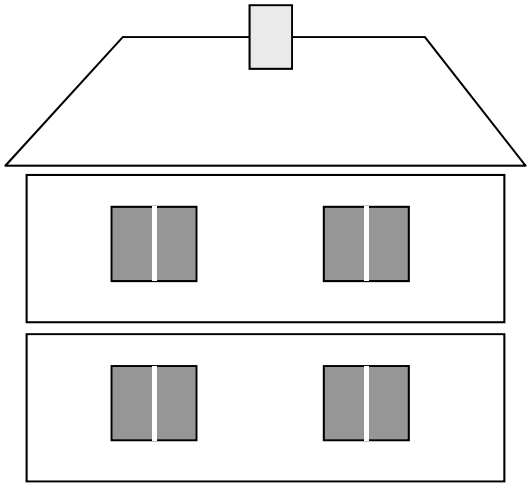


Nabízí se 3 způsoby reprezentace:

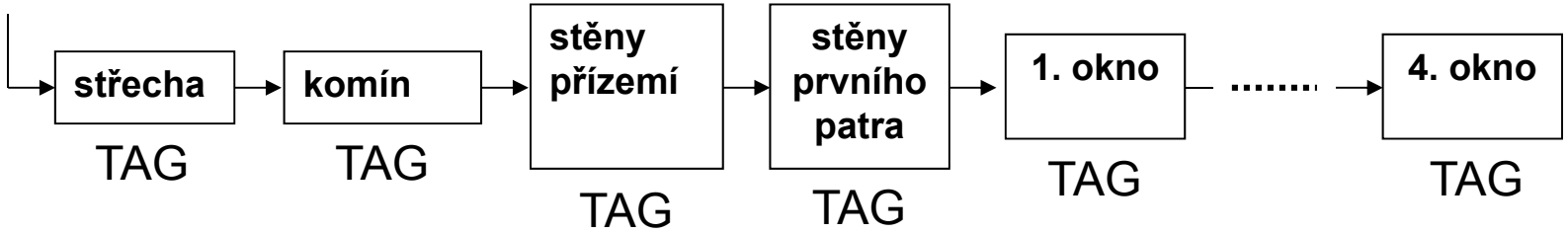
1. Posloupností segmentů (lineární datová struktura)
2. Hierarchický model – naivní strom
3. Hierarchický model – orientovaný acyklický graf

1. Lineární (nehierarchická) reprezentace

- objekt je reprezentován **posloupností segmentů** (lineární datová struktura)
- **segment** obsahuje definici grafických elementů a jejich atributů i transformací (př. levé okno v přízemí)



dům



- neexistuje informace o vzájemných vazbách objektů
  => snadno se manipuluje se segmenty,
  => ale těžko s komplexními strukturami (celé patro)

1. Lineární (nehierarchická) reprezentace

- Kdy použít lineární strukturu v programu?
- Pouze pro jednoduché objekty

```
void house(void) {        /* model celého domu */
   roof();                /* střešní segment */
   chimney();
   groundFloorWalls();
   firstFloorWalls();
   firstWindow();
   …
   fourthWindow();
}
```

2. a 3. Hierarchická reprezentace

- Vztahy mezi objekty modelu reprezentujeme **grafem,** kterému říkáme **GRAF SCÉNY ("scene graph")**

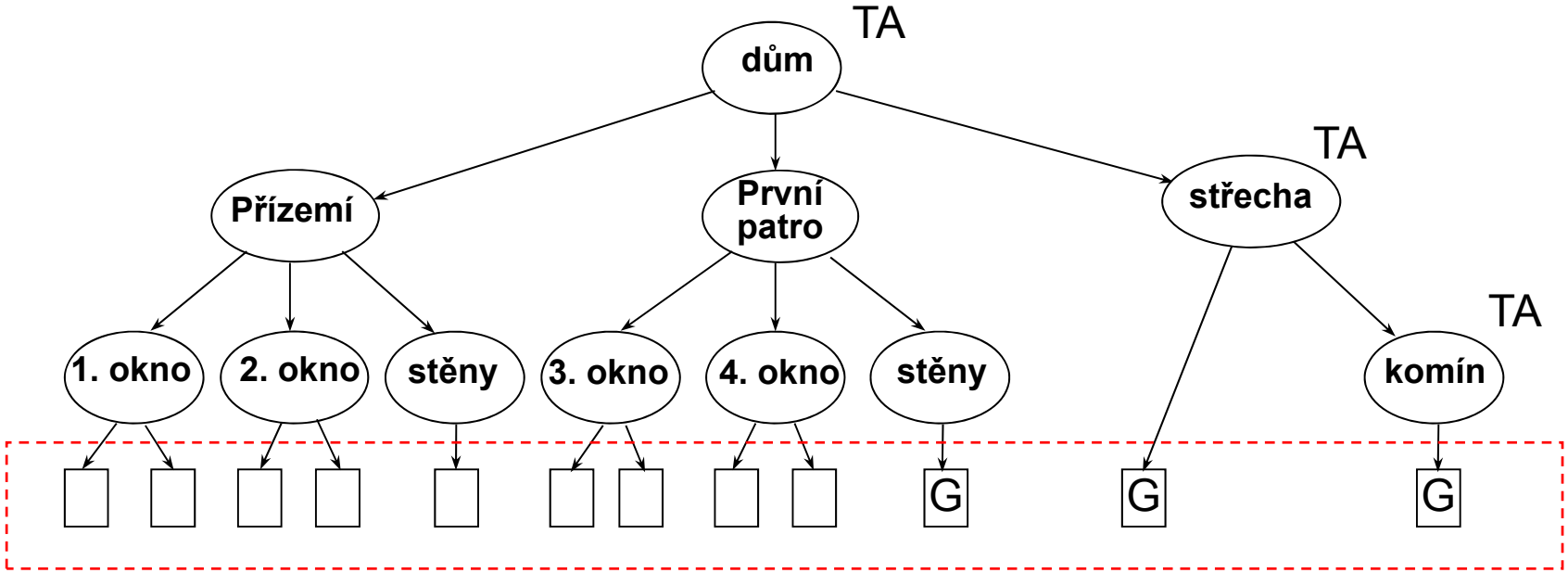
- Graf scény není konstruktem OpenGL, strukturu scény si v programu vytváříme sami.

- V grafu uložena (v závorce uvedeno ve které části grafu)
  - Geometrie G (listy),
  - transformace T
    atributy A (vnitřní uzly, hrany, listy)
  - odkazy na součásti (hrany)

- Vykreslení objektu = systematické procházení (traverzování) grafu
  - do hloubky či do šířky
  - v programu zvolit jeden způsob traverzování a neměnit ho
  - atributy se dědí, nebo jsou v uzlu předefinovány

2. Hierarchický model – naivní strom

- Jednoduchý graf je **kořenový strom** (orientovaný graf bez uzavřených cest a smyček, každý uzel kromě kořene má předchůdce-rodiče)
- **Vnitřní uzly** reprezentují nadřazené segmenty (detaily)
- **Listy** reprezentují grafická primitiva – opakují se



listy = grafická primitiva (+ atributy)

2. Hierarchický model – naivní strom

Jak zakódovat strom v programu?

- **vnitřní uzly** – ukládají transformace, atributy a odkazy na následníky
- **listy** – reprezentovány kreslícími funkcemi (geometrie)

Př. Napevno v programu:

```
void house(void) {          /* model celého domu */
    // (posunutí do počátku)
    groundFloor(); // včetně transformace pro přízemí
    firstFloor();  // včetně transformace pro 1. patro
    roof();        // včetně transformace pro střechu
}
```

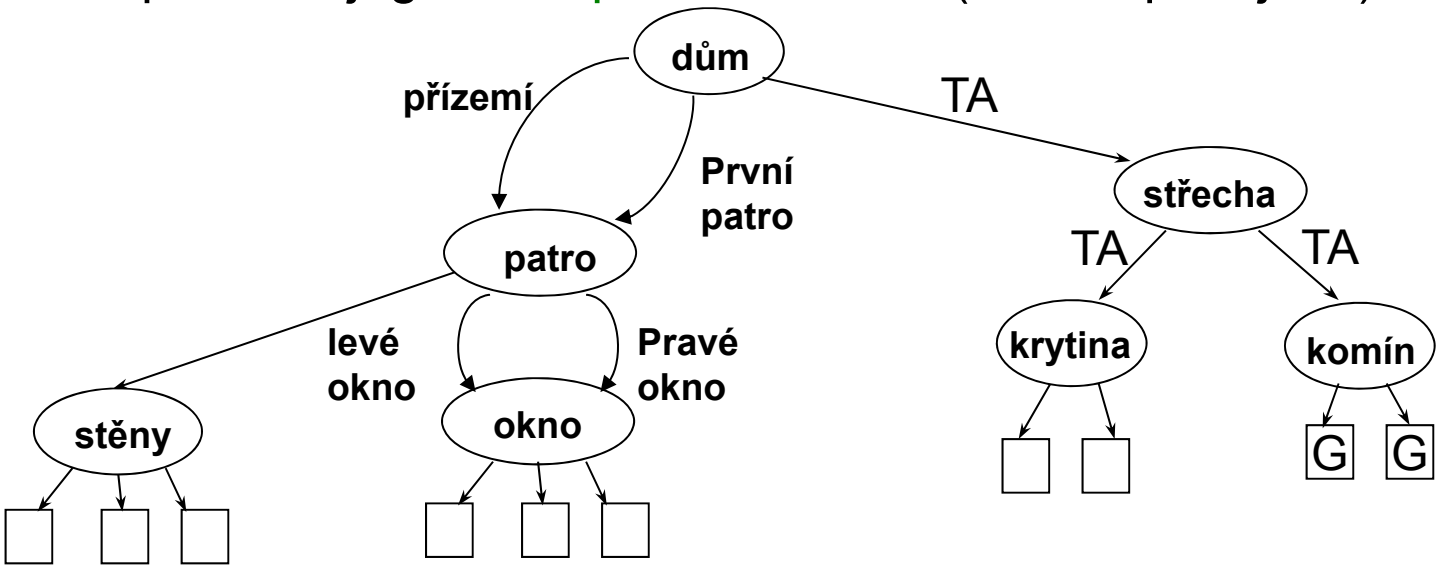
- Problémy reprezentace s použitím stromu:
  - opakuje se geometrie (listy)
    i stejné tvary (detaily) v různých polohách (vnitřní uzly)
  - transformace kódovány napevno v uzlech – obtížná manipulace
    (jeden celek (uzel) nejlze použít vícekrát na různých místech)

3. Orientovaný acyklický graf (DAG)

Též Collapsed tree (VRML)

Opakující se detaily se uloží jen jednou - (př. jedna definice komínu)

- **Vnitřní uzly –** reprezentují detaily dané úrovně modelu,
  – optimálně v počátku [0,0,0] (ale neopakují se)
- **Hrany –** reprezentují vazby (relace rodič-potomek)
  – nesou transformace, atributy
  (inkrementální změnu na cestě od rodiče k potomkovi)
- **Listy –** reprezentují grafická primitiva (ale neopakují se)



3. Orientovaný acyklický graf (DAG)

Jak zakódovat DAG v programu?

- **Vnitřní uzly** – ukládají odkazy na potomky v základní poloze
  případně „normalizační“ transformaci
  pozor na normály, viz. další lekce
- **Hrany** – transformace, atributy
- **Listy** – reprezentovány kreslícími funkcemi

Př. :

```
void house(void) {          /* model celého domu */
    /* nastav transformace pro přízemí */
    floor(); // v základní poloze
    /* nastav transformace pro první patro*/
    floor();                ;
    /* nastav transformace pro střechu */
    roof();
}
```

Reprezentace grafu

- specializovanou knihovnou – „grafem v paměti“ (OpenSceneGraph, OpenSG, OpenVRML, dcgiSceneGraph)
  - pro graf scény zvláštní třída a funkce
  - při vykreslení se traverzuje strom (transformují, volají kreslící procedury)
- skrytě
  - pomocí vlastní datové struktury reprezentující graf scény
  - pořadím volání vykreslovacích procedur
  - může být méně obecná a proto rychlejší

Hierarchická reprezentace - shrnutí

Výhody hierarchického modelování (DAG)

- hierarchické modely odrážejí strukturu objektů
- dobře se vytvářejí (top-down nebo bottom-up)
- dobře se editují a animují, dobře se manipuluje s podstrukturami (transformace jsou součástí hran)
- paměťově úsporná reprezentace modelu (bez redundance)

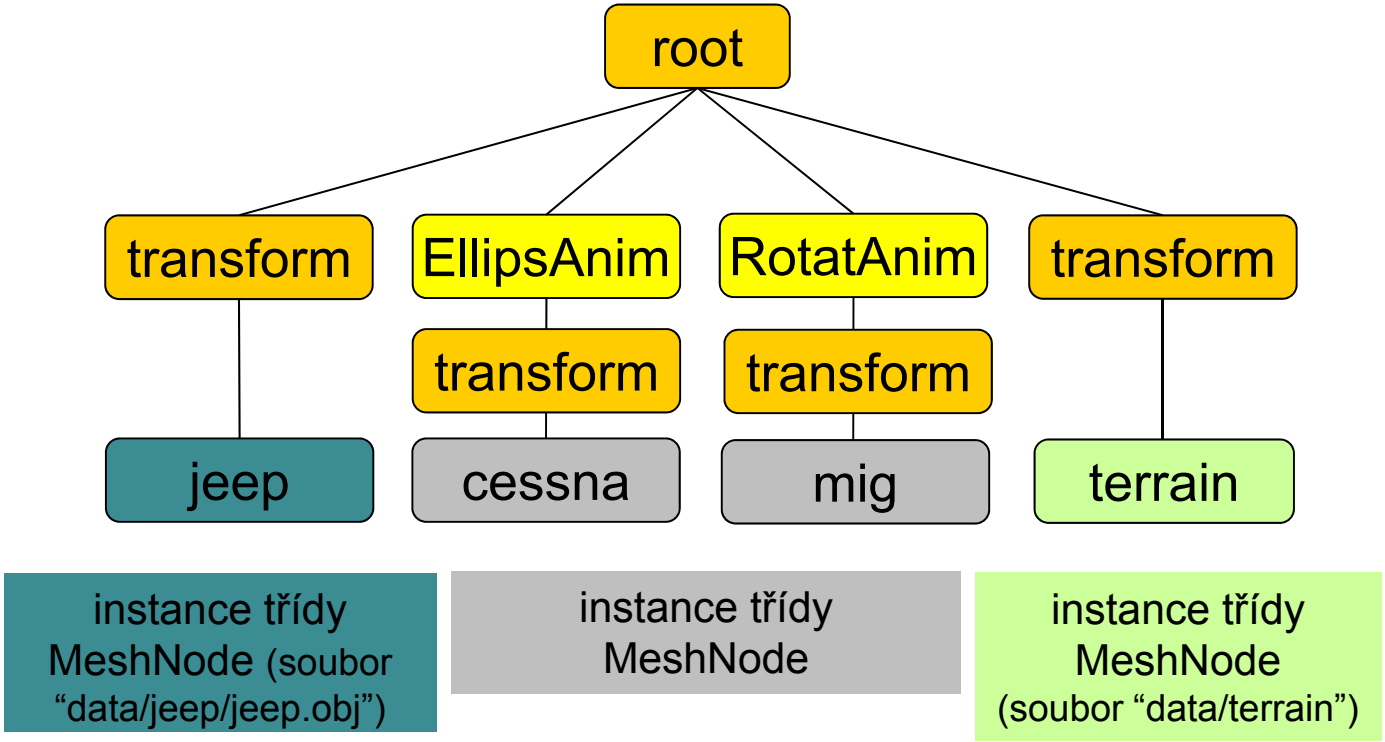
Atributy v grafech scény

POZOR na atributy při traverzování grafu scény

- atributy jsou stavové proměnné – proto zůstávají nastavené, dokud je nenastavíme jinak
  - to platí pro některé implementace
  - výsledek pak může záviset na pořadí traverzace
  - v našich příkladech to neplatí – na pořadí traverzace nezáleží
- pro animaci grafu (dynamické nastavování transformací) se používá několik postupů
- probereme jeden z možných způsobů, použitý pro náš graf scény

Vlastní graf scény

Jednoduchý graf scény pro účely předmětu PGR, viz cvičení.



Vlastní graf scény

Typy uzlů

- **SceneNode** – bázový typ
  – obsahuje vektor odkazů na potomky
  - **TransformNode** – ukládá transformační matici
    – vnitřní uzel grafu
  - **RotationAnimNode** – rotace kolem osy pro animaci
    – vnitřní uzel grafu
  - **MeshNode** – ukládá vykreslitelnou geometrii
    – list grafu
    - TerrainNode – terén vygenerovaný programem terragen
    - OBJNode – objekt ve formátu .obj
- **Metody** – update / pro animace
  – addChildNode/deleteChildNode - správa hierarchie

scene

transform

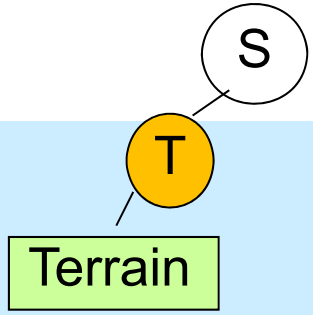
RotatAnim

terrain

OBJ

Vlastní graf scény - dcgiSceneGraph

Ruční sestavení grafu scény 1



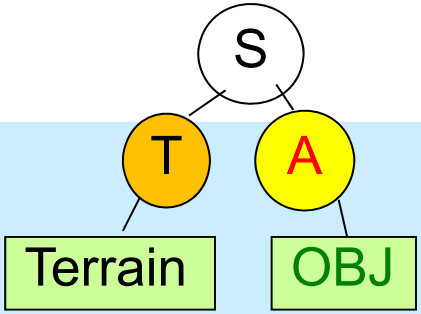

```
void init() {
  root_node = new SceneNode( "root" );

  // terrain transform node
  CTransformNode* transform_p =
                  new CTransformNode( root_node );
  // terrain node
  CTerrainNode *terrain_p =
                  new CTerrainNode( transform_p );
  terrain_p->load("terrain");

…
```

Vlastní graf scény - dcgiSceneGraph

Ruční sestavení grafu scény 2




```
....
....
// Model bude rotovat kolem osy procházející počátkem
souřadnic
 RotationAnimNode * prot
                = new RotationAnimNode("py-rot2", root_node);
 prot->setAxis(Vec3f(1, 0, 0));
 prot->setSpeed(M_PI / 10);

  // Nyní nahraj model z formátu OBJ a učiň ho potomkem uzlu
s definicí
  // animované rotace, rodič je správně nastaven
  MeshNode *mesh = new MeshNode("cessna.obj", prot);
  mesh->loadMesh(); // load the data
  …
```

Vlastní graf scény - dcgiSceneGraph

Vykreslení grafu scény

```
void functionDraw(void) {
  // vymaž obrazovku
  glClearColor(0.3f, 0.3f, 0.3f, 1.0f);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  // nastavení matice projekce = perspektivy
  glm::mat4 projection = Matrix4f::Perspective(M_PI / 4,
                         aspect_ratio, 0.1f, 100);
  // nastavení modelovací a pohledovou matici
  Matrix4f view = Matrix4f::Identity(); Matrix4f model=
  SendMatricesToShaders(projection, view, model)
  // vykresli celou scénu = traverzuj graf scény
  rootNode_p->draw();
}
....   glutSwapBuffers();
```

Vlastní graf scény - dcgiSceneGraph

Jednoduchá aktualizace transformace pomocí časovače

```
void FuncTimerCallback(int)
{
  double timed = 0.001 *
   (double)glutGet(GLUT_ELAPSED_TIME); // v milisecundách

   // animuj scénu pro zadaný čas
   if (root_node)
      root_node->update(timed);

   // nastav další volání časovače pro příště
   glutTimerFunc(33, FuncTimerCallback, 0);
   // vyvolej překreslení obrazu
   glutPostRedisplay();
}
```

Vlastní graf scény - dcgiSceneGraph

Příklad: inkrementální rotace, viz třída `RotationAnimNode{ …`

```
TransformNode * ptrans2 = new TransformNode("py-trans2",
root_node);
  ptrans2->translate(Vec3f(0.5, 0, -3));
  ptrans2->scale(Vec3f(0.05, 0.05, 0.05));

  RotationAnimNode * prot2 = new RotationAnimNode(
                                "py-rot2", ptrans2);
  prot2->setAxis(Vec3f(1, 0, 0));
  prot2->setSpeed(M_PI / 10); // v radiánech za sekundu
  MeshNode * mesh1 = new MeshNode("data/cessna.obj",prot2);
  mesh1->loadMesh();
```

Skládání transformací v grafu scény

- Transformace se při postupu shora dolů skládají zleva doprava
  a) Vyhodnocují se během vykreslování
     výhodné použít zásobník: dolů – push / nahoru – pop
  b) Vyhodnotí se v samostatném průchodu – update()
     složené transformace se uloží v hranách
     (takto je to v dcgiSceneGraphu)