

A Generic Rendering System

Jurgen Dollner, Klaus Hinrichs

Martin Sobek, 2003

A Generic Rendering System – Motivace

- není možné vytvořit nový renderovací systém „od nuly“
- cílem bylo navrhnout systém, který sjednocuje dobré vlastnosti dosud existujících grafických knihoven a poskytuje vhodný interface, který nepotlačuje, ale podporuje jejich vlastnosti

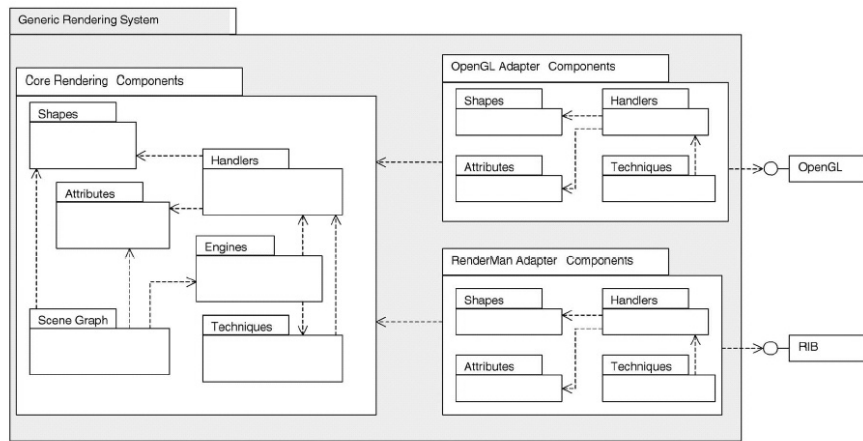
Renderovací systémy

- White-box rendering systems
 - dokumentují svůj design a umožňují přístup k implementaci
- Black-box rendering systems
 - skrývají svou architekturu a jsou použitelné skrz dobře popsaný interface
- Gray-box rendering systems
 - „něco mezi“ – je možné systém rozšiřovat děděním tříd

Omezení existujících architektur

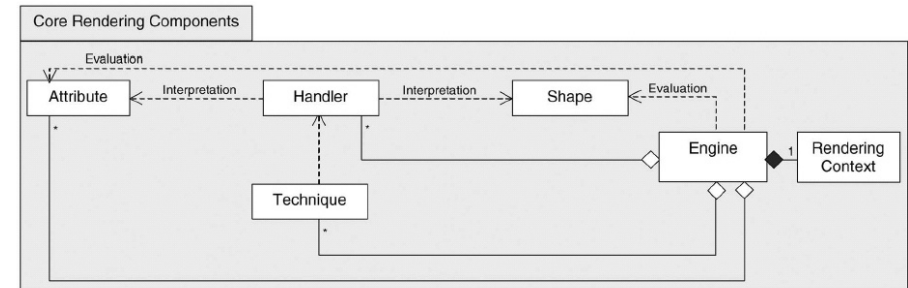
- WB a GB systémy nabízejí objektový model, který je podřízený implementaci
 - omezená portabilita
 - omezená rozšiřitelnost

Hlavní komponenty a závislosti



Main software packages of the generic rendering system.

Vztahy mezi komponentami jádra systému



Uses associations and part-of associations between core rendering components.

Attributes a Shapes

- Shapes jsou objekty, které mají nějaký význam z pohledu cílového média (geometrie nebo i zvuk)
 - typicky mohou obsahovat popis základních geometrických objektů
 - neobsahují žádnou „znalost“ (algoritmus) ale pouze popis
- Attributes fungují jako modifikátory
 - barva, textura, material, transformace, atd.
 - rovněž neobsahují žádný algoritmus
 - monoatributy (např. material) x polyatributy (např. světlo)

Handlers

- reprezentují algoritmus zpracování atributů nebo tvarů
- atributy a tvary „nevědí“, jaký handler bude použit pro jejich zpracování
- každý handler je zodpovědný za určitý atribut nebo tvar – tzv. target a umí s ním provádět určitou funkcionalitu – tzv. service

Handlers (2)

- odděluje se reprezentace a data od algoritmu – koule „se neumí nakreslit“
- handlers mohou být dynamicky přidávány do engineu – přesouvá se vazba funkcionality s objektem z compile-time na run-time

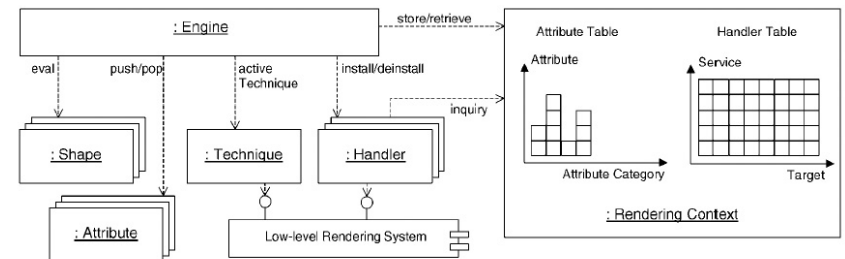
Techniques

- určují postup vyhodnocování sekvence komponent
- rozhodují, které handlers se použijí pro zpracování atributů a tvarů poslaných do engineu a organizují spouštění handlerů
- typický vyhodnocovací proces je syntéza obrazu – techniques hledají handlers, které mají schopnost aplikovat atributy a nakreslit tvary

Engines

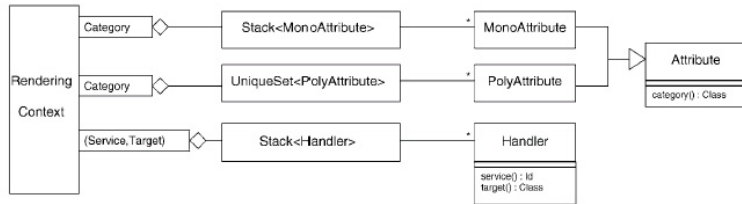
- spravují rendering context
- slouží jako interface pro komunikaci s „vnějším světem“
- uzly scény zasílají objekty, které obsahují, právě engineu
- engine slouží jako návštěvník uzlů v grafu scény

Diagram úkolů engineu



Tasks of an engine to process rendering components.

Rendering context



Class model of the rendering context.

Strategie vyhodnocování

- engine neimplementuje žádnou strategii vyhodnocování – ta je řízena technikou
- engine slouží pouze jako interface s následujícími metodami:
 - push/pop monoatributů
 - přidat/odebrat polyatribut
 - instalovat/odinstalovat handler
 - vyhodnotit scénu
 - každý engine spolupracuje s právě jednou technikou (může být změněna run-time)

Strategie vyhodnocování (2)

- techniky závislé na low-level renderovacím systému
 - renderovací technika
 - mapuje atributy a tvary na odpovídající konstrukce low-level renderovacího systému (OpenGL, RenderMan, atd.)
- techniky nezávislé na low-level r.s.
 - ray-picking
 - collision detection
 - attribute search
 - image information

Strategie vyhodnocování (3)

- do enginu je poslán atribut (podobně pro tvar)
- engine uloží atribut do rendering context
- engine přenechá zpracování atributu aktivní technice
- technika vyhledá příslušný handler
 - např. pro atribut material se hledá handler pro kategorii „material“ a službu „attribute deployment“, jestli-že je handler nalezen, mapuje vlastnosti materiálu na příslušné funkce low-level r.s.

Strategie vyhodnocování (4)

- jestliže nebylo možné nalézt odpovídající handler, pokouší se nalézt simplifier
- nepodaří-li se nalézt ani simplifier, rekurzivně se proces opakuje, použije se ovšem třída předka
- v případě že se nepodaří najít žádný handler, vyvolá se chybový stav

Tajný kód

```
void OpenGLRendering::execEval(Shape s, Engine e) {
    if (handleShape(s, e, OPENGLPAINTING) == false) {
        error handling for missing service
    }
}

bool Technique::handleShape(Shape s, Engine e, Id service) {
    Class target = s.Class();
    while (target != NULL) {
        Handler h = e.context().handlerTable(service, target);
        if (h != NULL) {
            h.exec(s, e);
            return true;
        }
    }
}

h = e.context().handlerTable(SIMPLIFICATION, target);
if (h != NULL) {
    h.exec(s, e);
    return true;
}
// no success, find handlers based on //parent class
target = target.parentClass();
}
// no suitable handlers found
return false;
}
```

Základní služby

- Shape simplification
- Shape painting
- Attribute simplification
- Attribute painting
- Ray intersection
- Handlery a techniky nejsou z vnějšího světa běžně viditelné – konstruktor enginu nastaví rendering context na správné handlery

Ukázka použití

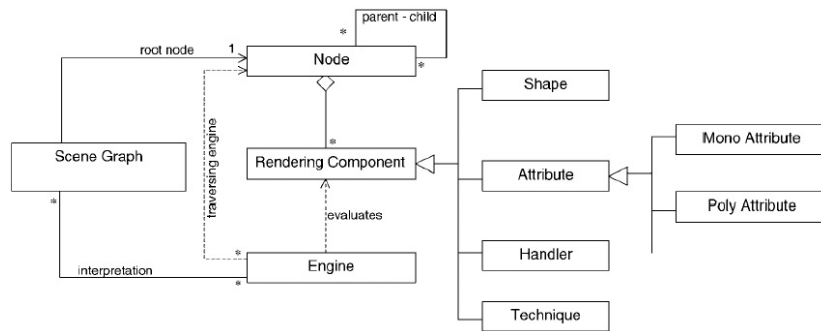
```
void example(Engine e) {
    Material mat = new Material(...);
    Rotation rot = new Rotation(...);
    Torus ts = new Torus(...);

    e.push(mat, mat.category());
    e.push(rot, rot.category());
    e.eval(ts);
    e.pop(mat.category());
    e.pop(rot.category());
}

void example(Engine e) {
    Torus ts = new Torus(...);
    TorusPainterOpenGL tspainter =
        new TorusPainterGL();

    e.install(tspainter);
    e.eval(ts);
    e.deinstall(tspainter);
}
}
```

Scene graph



• 2 druhy procházení grafem

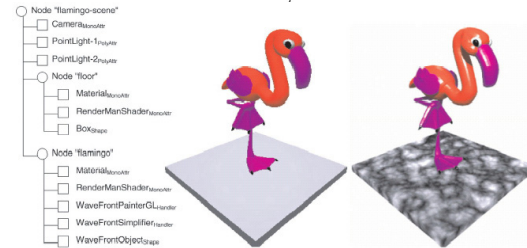
- evaluation
- inspection (pouze zjišťuje strukturu)

Node

- atributy uložené v uzlu ovlivňují pouze potomky a nikoliv sourozence

```

class Node {
private: List<Object> contentObjects;
public:
void evaluate(e : Engine) {
    unapply(apply(contentObjects,e),e);
}
void inspect(v : Visitor) {
    for each c in contentObjects do {
        v.explore(c);
        if (c is a node) c.inspect(v);
    }
};
}
Stack apply(L : List<Object>, e : Engine) {
    S : Stack = ();
    for each c in L {
        if (c is a node) c.evaluate(e);
        else if (c is a shape) e.eval(c);
        else {
            S.push(c);
            if (c is a mono attribute)
                e.push(c, c.category());
            else if (c is a poly attribute) { e.add(c, c.category()); }
            else if (c is a handler) e.install(c);
        }
    }
    return S;
}
    
```



Vlastnosti grafu scény

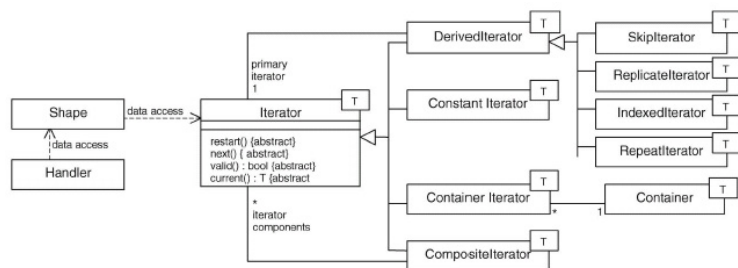
- oddělení tvarů a atributů
- oddělení deklarace od implementace
- oddělení struktury od obsahu
- graf scény je chápán jako šablona, která je interpretována konkrétním enginem

Vytváření nových tříd

- je nutné, aby by systém rozšiřitelný
- vždy je nutné vytvořit třídu tvaru
- v jednodušším případě stačí napsat handler – simplifier
- lepší je napsat i specializované handlers pro konkrétní služby

Iterátory

- data tvarů jsou obvykle reprezentovány různými druhy polí, které mohou mít různou strukturu
- proto se zavádí iterátory
 - základní a high-level



Vlastnosti atributů

- druhy atributů
 - surface shading attributes (color, texture, ...)
 - scene attributes (fog, antialiasing, bg, ...)
 - light source attributes
 - geometry attributes (transformations, lod, ...)
- počet kategorií atributů není omezen
- atributy mohou být buďto čteny při zpracování tvaru, nebo mohou nastavit „stav“ low-level r.s.

Víceprůchodové renderování

- používá se pro dosažení speciálních efektů
- je realizováno technikou, které má následující interface:
 - start, stop
 - needsPass, nextPass
 - preparePass, finishPass
 - + zděděné ze základní třídy Technique

Aplikace v praxi

- Virtual rendering system (VRS) je implementací Generic Rendering Systému v jazyce C++
- umí převést veškeré zabudované tvary na síť trojúhelníků, takže je snadné napsat low-level adapter
- používá OpenGL pro real-time rendering a RenderMan pro vysoce kvalitní renderování

Závěr

- GRS reprezentuje věcný přístup k integraci různých systémů
- je snadno rozšiřitelný a lze rovněž snadno psát aplikace