

Seznam úloh do předmětu Počítačová grafika II

(aktualizace 28. 4. 2021)

Ukončení předmětu proběhne udělením klasifikovaného zápočtu na dálku za odevzdaný projekt. Hodnoceny budou jednotlivé úlohy z níže uvedeného seznamu. U jednotlivých úloh jsou uvedeny maximální počty bodů, které lze za vyřešení dané úlohy získat. Předpokládá se, že součástí projektu budou zdrojové kódy, obrázky s dosaženými výsledky, stručný popis odevzdávaných materiálů a případně další součásti, pokud nejsou odkazovány na stránkách předmětu.

Nejzazším termínem pro zaslání projektu (mailem) je 17. 5. 2020 včetně. V případě 5. ročníku je termín 3. 5. 2020 včetně.

1. úloha (slidy 19 – 34) 15 bodů

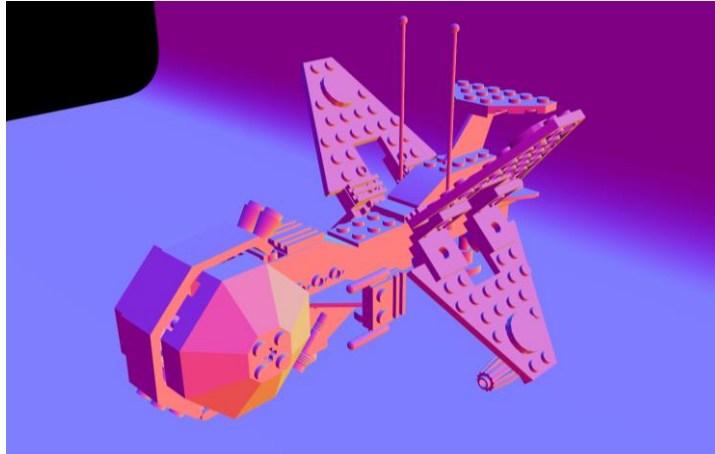
Implementace kamery dle přednášek. Ideálně vytvořte v třídě Camera metodu Update, která provede výpočet všech potřebných matic (V , P) na základě aktuálních parametrů kamery. Jednotlivé matice a jejich kombinace (MVP pro transformaci vertexů do Clip Space, MV pro transformaci vertexů do Eye Space, MVn pro transformaci normál do Eye Space apod.) pak můžete podle potřeby předávat (přes ukazatel na první prvek – metoda data) ve vykreslovací smyčce pomocí funkce SetMatrix4x4 (glutils.cpp) aktuálnímu shaderu. Zároveň ve vykreslovací smyčce zajistěte, aby se kamera vhodným způsobem pohybovala.

```
void SetMatrix4x4( const GLuint program, const GLfloat * data, const char * matrix_name )
{
    const GLint location = glGetUniformLocation( program, matrix_name );

    if ( location == -1 )
    {
        printf( "Matrix '%s' not found in active shader.\n", matrix_name );
    }
    else
    {
        glUniformMatrix4fv( location, 1, GL_TRUE, data );
    }
}
```

2. úloha (slidy 131 – 140) 15 bodů

Načtení modelu z OBJ souboru včetně materiálů z MTL. Načítač OBJ a MTL je součástí šablony. Stačí podle ukázek z funkce tutorial_1 (tutorials.cpp) doimplementovat plnění VAO a VBO objektů načtenými daty. Funkčnost lze otestovat např. pomocí jednoduchého normálového shaderu. Výsledek by měl vypadat jako na přiloženém obrázku.



Z hlediska organizace kódu, doporučuji vytvořit si třídu Rasterizer, která bude obsahovat implementaci této a dalších úloh. Struktura implementovaných metod může být následující:

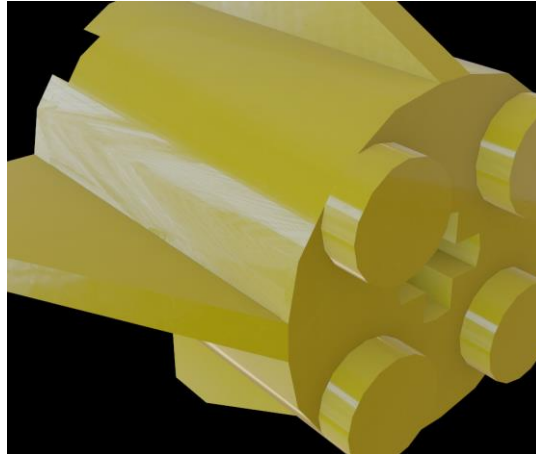
```
int test( const std::string & file_name )
{
    Rasterizer rasterizer( 640, 480, deg2rad( 45.0f ), Vector3( 200, 300, 400 ),
Vector3( 0, 0, 30 ), 1.0f, 1000.0f );
    rasterizer.InitDevice(); // inicializace OpenGL kontextu
    rasterizer.InitPrograms(); // inicializace VS a FS shaderů (programu)
    rasterizer.LoadScene( file_name ); // načtení geometrie scény
    rasterizer.InitBuffers(); // inicializace VAO a VBO
    rasterizer.InitMaterials( 0 ); // inicializace SSBO pro uložení materiálů
    // následuje inicializace předpočítaných map pro Cook-Torrance GGX shader
    rasterizer.InitIrradianceMap( "../../data/lebombo_irradiance_map.exr" );
    rasterizer.InitPrefilteredEnvMap( {
        "../../data/lebombo_prefiltered_env_map_001_2048.exr",
        "../../data/lebombo_prefiltered_env_map_010_1024.exr",
        "../../data/lebombo_prefiltered_env_map_100_512.exr",
        "../../data/lebombo_prefiltered_env_map_250_256.exr",
        "../../data/lebombo_prefiltered_env_map_500_128.exr",
        "../../data/lebombo_prefiltered_env_map_750_64.exr",
        "../../data/lebombo_prefiltered_env_map_999_32.exr" } );
    rasterizer.InitGGXIntegrMap( "../../data/brdf_integration_map_ct_ggx.png" );
    rasterizer.MainLoop(); // renderovací smyčka

    return EXIT_SUCCESS;
}
```

3. úloha (slidy 44 – 72) 20 bodů (+10 bonusových bodů)

Implementace PBR shaderu dle přednášek. Veškeré předpočítané mapy jsou k dispozici na webu předmětu včetně jednoduchého modelu s PBR materiálem. Při vlastní implementaci předpočítání jednotlivých map získáte dalších 10 bodů.

Výsledek by při použití jednoduchého modelu kostky mohl vypadat následovně.



4. úloha (slidy 75 – 82) 15 bodů

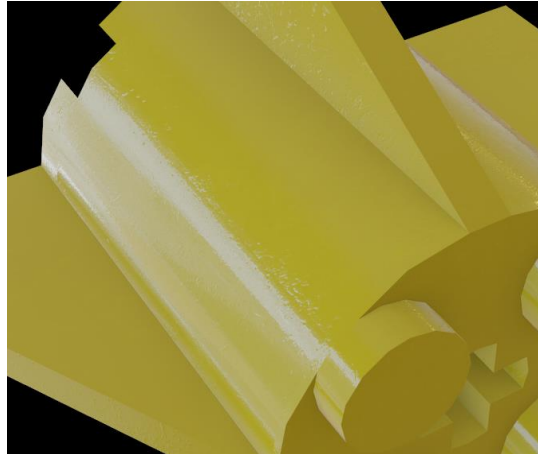
Implementace podpory pro normálovou mapu dle přednášek. Pro vyřešení této úlohy je potřeba (např.) rozšířit konstruktor třídy Triangle o výpočet tečných vektorů v každém vrcholu trojúhelníka.

```
Triangle::Triangle( const Vertex & v0, const Vertex & v1, const Vertex & v2, Surface *
surface )
{
    vertices_[0] = v0;
    vertices_[1] = v1;
    vertices_[2] = v2;

    // TODO TBN
    const Vector3 e1 = v1.position - v0.position;
    const Vector3 e2 = v2.position - v0.position;
    // t = ?, b = ?
    vertices_[0].tangent = Tangent( t, b, v0.normal );
    vertices_[1].tangent = Tangent( t, b, v1.normal );
    vertices_[2].tangent = Tangent( t, b, v2.normal );
}

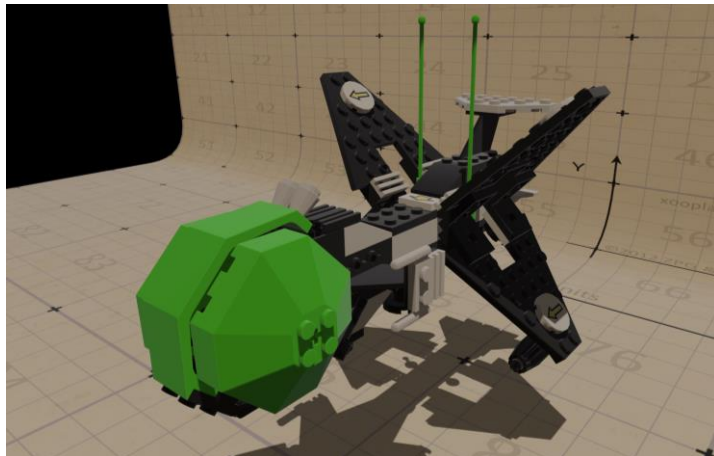
// pomocná funkce pro výpočet tečného vektoru daného vrcholu trojúhelníka
// vstupem je tangenta, bitangenta a normála před GS ortogonalizačním procesem
// výstupem je tangenta po provedení GS ortogonalizačního procesu
Vector3 Tangent( const Vector3 & t, const Vector3 & b, const Vector3 & n )
{
    return tc;
}
```

Výsledek by při použití jednoduchého modelu kostky mohl vypadat následovně. Všimněte si odrazů ovlivněných nerovnostmi na povrchu kostky simulovanými pomocí normálové/bump mapy. V předchozím případě se povrch jevil jako zcela hladký.



5. úloha (slidy 129-142) 15 bodů

Implementace stínů pomocí shadow mappingu. Postačuje jeden zdroj světla typu spot light (středové promítání) nebo directional light (rovnoběžné promítání). Nemusíte implementovat všesměrové bodové zdroje (omni light), kde by bylo nutné vytvořit 6 zdrojů typu spot light a výsledná mapa stínů by byla reprezentována jako cube mapa.



6. úloha (slidy 159-187) 20 bodů

Implementace stínů pomocí stencil bufferu. Postačuje jeden zdroj světla typu spot light (středové promítání). Také postačí, když implementace bude fungovat nad zjednodušeným modelem.

