Kurz LSL skriptování

Shiny Iceberg

2009

součást cyklu
Nejsme jelita

Kurz LSL skriptování

Shiny Iceberg
v Second Life od roku 2006
shiny.iceberg@virtualmagazine.cz

Aktuální projekty

virtualmagazine.cz
Urbanica, Shinyland
Bwindi Orphans
cyklus Nejsme jelita

Organizační body

Průběh lekce

- bude trvat zhruba 90-120 minut
- pokud nekladete dotazy, vypněte si mikrofon
- příklady si klidně zkoušejte přímo v hledišti

Vaše otázky

- můžete se ptát na konci každého snímku
  nebo na konci celé přednášky
- dotazy mohou být přes voice nebo IM
- při psaní IM z posledních řad použijte Shout

To nejdůležitější z minula

- llGetKey / llGetOwner
- llGetObjectName / llGetObjectDesc
- llSetObjectName / llSetObjectDesc
- llGetScale / llSetScale
- llGetAlpha / llSetAlpha
- llGetColor / llSetColor
- llGetTexture / llSetTexture
- llSetText / llSetSitText / llSetTouchText
- llSetPrimitiveParams

Plán přednášky

1. Fungování a struktura skriptu
2. Z čeho se skládá skript
3. Vlastnosti objektu
4. Pohyb objektu
5. Pose bally
6. Komunikace skriptu
7. Inventory objektu
8. Detektory
9. Particles
10. Příklady a dokumentace

Hmotné a nehmotné objekty

- Objekty s příznakem PHYSICAL se chovají podle fyzikálních zákonů (hmotnost v lindogramech, síla, zrychlení), chování těchto objěktů řídí systém zvaný Havoc
- Objekty bez toho příznaku se chovají jako nehmotné objekty - nepůsobí na ně gravitace, mohou se prolínat atd.
- Každá kategorie objektů potřebuje jiné funkce pro pohyb a rotaci
- Speciálním případem PHYSICAL objektů jsou vozidla
  http://lslwiki.net/lslwiki/wakka.php?wakka=vehicles

Úvod k probíraným funkcím

- V každé lekci probereme jen vybrané funkce, kompletní seznam na LSL wiki je mnohem obsáhlejší

http://www.lslwiki.net/lslwiki/

- Až na předem řečené výjimky mohou být všechny příklady testovány na základní rezznuté kostce

Seznam probíraných funkcí

- llGetStatus / llSetStatus
- llGetPos / llGetLocalPos / llGetRootPosition
- llSetPos
- llGetRot / llGetLocalRot / llGetRootRotation
- llSetRot
- llSetPrimitiveParams
- llTargetOmega
- llRot2Up / llRot2Fwd / llRot2Left
- llGetMass / llGetVel
- llGetForce / llSetForce
- llApplyImpulse

Variable *vector*

- jde o tři čísla typu *float* spojené dohromady ve tvaru < x , y , z > a tato trojice umožňuje vyjádřit například polohu, rychlost, barvu, měřítko atd.

- pomocí tečkové notace mohu měnit jednotlivé složky vektoru, viz příklad

```
vector Poloha = <122.7, 98.8, 24.5>;

Poloha.z = 28.1;
```

Variable *rotation*

- rotace není v LSL vyjádřena pomocí klasického otočení podle os X, Y a Z, ale pomocí takzvaných kvaternionů - tvoří ji 4 hodnoty <X, Y, Z, S>

- naštěstí lze klasickou XYZ rotaci jednoduše převádět na a z kvaternionů pomocí dvou funkcí llEuler2Rot a llRot2Euler

Převod *rotation* na stupně

kvaterniony ⇔ radiány ⇔ stupně

```
// aktualni rotace objektu
rotation Kvaternion = llGetRot();
vector RotaceRadiany = llRot2Euler(Kvaternion);
vector RotaceStupne = RotaceRadiany * RAD_TO_DEG;

// rotace o 45 stupnu podle osy Z
vector RotaceStupne = <0,0,45>;
vector RotaceRadiany = RotaceStupne * DEG_TO_RAD;
rotation Kvaternion = llEuler2Rot(RotaceRadiany);
```

integer llGetStatus(integer *flag*);

VSTUP: typ vlastnosti, u které se zjišťuje status, pro zjištění vlastnosti PHYSICAL se použije konstanta *STATUS_PHYSICS*

VÝSTUP: TRUE (1) nebo FALSE (0)

```
default
{
    state_entry()
    {
        // Do promenne JeHmotny se bud ulozi 0 nebo 1
        integer JeHmotny = llGetStatus(STATUS_PHYSICS);
    }
}
```

```
llSetStatus(integer flag, integer AnoNe);
```

VSTUP:

- ***flag:*** typ vlastnosti, u které se nastavuje status, pro nastavení vlastnosti PHYSICAL se použije konstanta *STATUS_PHYSICS*

- ***AnoNe:*** TRUE (1) nebo FALSE (0)

VÝSTUP: nic

```
vector llGetPos();
```

VSTUP: nic

VÝSTUP: souřadnice aktuální polohy primu

POZNÁMKA:

- pokud se funkce volá z child primu, vrátí souřadnice child primu (nikoliv root primu) a v absolutních souřadnicích regionu (nikoliv relativně k root primu)
- pokud se volá z root primu attachmentu, vrátí polohu avatara

```
vector llGetLocalPos();
```

VSTUP: nic

VÝSTUP: souřadnice relativní polohy child primu vůči root primu, při použití v root primu funguje stejně jako llGetPos

---

```
vector llGetRootPosition();
```

VSTUP: nic

VÝSTUP: absolutní souřadnice root primu

```
llSetPos(vector Pozice);
```

VSTUP: nové souřadnice objektu

VÝSTUP: nic

POZNÁMKA:

- pokud se volá z root primu, jsou souřadnice chápány jako absolutní tj. vzhledem k regionu, pokud se funkce volá z child primu, jsou souřadnic brány jako relativní tj. vzhledem k root primu
- funkce nedokáže přesunout objekt pod zem, výše než 4096 metrů a maximální délka posunu je 10 metrů, naopak lze posunout objekt do sousedního regionu
- používat jen na non-PHYSICAL objekty

Příklad na multi-posun objektu

```
// O kolik metru ma objekt vzletnout
float PosunNahoru = 25.0;
vector PuvodniPoloha;

default
{
    touch_start(integer total_number)
    {
        PuvodniPoloha = llGetPos();
        vector NovaPoloha = PuvodniPoloha + <0.0,0.0,PosunNahoru>;

        do
        {
            llSetPos(NovaPoloha);
        } while ( llVecDist (llGetPos(), NovaPoloha) > 0.001);

        llSleep(2.0);

        do
        {
            llSetPos(PuvodniPoloha);
        } while ( llVecDist (llGetPos(), PuvodniPoloha) > 0.001);

    }
}
```

Eventy spojené s pohybem

moving_start
aktivuje se v okamžiku, kdy se objekt začne pohybovat anebo při vstupu do nového simu

moving_end
aktivuje se v okamžiku, kdy se objekt přestane pohybovat anebo při opuštění simu

Eventy spojené s pohybem - příklad

Přesun objektu v Edit módu tyto eventy neaktivuje, proto je vhodné testovat příklad na PHYSICAL objektu.

```
vector start;

default
{
    moving_start()
    {
        start = llGetPos();
        // Pridano, aby moving_end nenaskocil moc rychle
        llSleep(1.0);
    }

    moving_end()
    {
        // llVecDist spocita vzdalenost dvou mist (vektoru)
        float vzdalenost = llVecDist(start, llGetPos());
        llOwnerSay("Posunuto o " + (string)vzdalenost + " metru.");
    }
}
```

```
rotation llGetRot();
```

VSTUP: nic

VÝSTUP: aktuální absolutní rotace primu

POZNÁMKA:

- pokud se funkce volá z child primu, vrátí rotaci child primu (nikoliv root primu) a v absolutních osách regionu (nikoliv relativně k root primu)
- pokud se volá z root primu attachmentu, vrátí rotaci avatara

```
rotation llGetLocalRot();
```

VSTUP: nic

VÝSTUP: relativní rotace child primu vůči root primu, při použití v root primu funguje stejně jako llGetRot

---

```
rotation llGetRootRotation();
```

VSTUP: nic

VÝSTUP: vrací absolutní rotaci root primu

`llSetRot(rotation Rotace);`

VSTUP: nová rotace celého objektu vůči osám regionu

VÝSTUP: nic

`llSetLocalRot(rotation Rotace);`

VSTUP: nová rotace child primu vůči root primu

VÝSTUP: nic

POZNÁMKA: LSLwiki stránky popisují do detailu celou problematiku rotace u linksetů a attachmentů

llSetPrimitiveParams

- komplexní funkce, která umožňuje nastavit veškeré vlastnosti primu
- umožňuje nastavit několik parametrů najednou, např. rotaci a polohu v jednom kroku
- funkce samotná je jednoduchá, komplikované jsou vstupní parametry, viz http://lslwiki.net/lslwiki/wakka.php?wakka=llSetPrimitiveParams
- dnes probereme pouze část parametrů, které se týkají dříve probíraných vlastností

`llSetPrimitiveParams(list` *`parametry`*`);`

VSTUP:

- neuspořádaný seznam požadovaných parametrů, vždy ve tvaru
  - konstanta označující druh změny
  - parametry pro daný druh změny

VÝSTUP: nic

```
default
{
    state_entry()
    {
        llSetPrimitiveParams([
            PRIM_PHYSICS, TRUE,
            PRIM_POSITION, <10.0, 15.0, 20.0>
        ]);
    }
}
```

Konstanty pro llSetPrimitiveParams

PRIM_PHYSICS
`integer` ***`ano_ne`***
Poznámka: *ano_ne* může být 0 nebo 1, resp. FALSE nebo TRUE

PRIM_POSITION
`vector` ***`souradnice`***

PRIM_ROTATION
`rotation` ***`rotace`***

```
llTargetOmega
(vector osa, float rychlost, float sila);
```

VSTUP:

- ***osa:*** vektor osy, podle které bude objekt rotovat
- ***rychlost:*** v radiánech za vteřinu (360° ~ 6.28 rad)
- ***sila:*** síla rotace, má smysl jen pro PHYSICAL objekty, ale musí být nenulová i pro non-PHYSICAL

VÝSTUP: nic

llSetTargetOmega - poznámky

- pro PHYSICAL objekty jde o plynulou server-side rotaci, ale současná implementace nefunguje vždy dobře

- funkce je naopak široce použitelná pro non-PHYSICAL objekty, kdy běží client-side (nezatěžuje sim), ale je potřeba počítat s několika fakty:
    - rotace je jen vizuální, nikoliv skutečná - objekt nemění svou skutečnou rotaci, sice ukazuje v edit menu, ale llGetRot neukáže změny
    - rotace se u každého avatara zobrazuje jinak
    - při editaci se rotace objektu vizuálně zresetuje
    - rotace se zastaví zavoláním llTargetOmega(ZERO_VECTOR, 0, 0);

```
vector llRot2Up (rotation rotace);      1
vector llRot2Fwd (rotation rotace);     2
vector llRot2Left (rotation rotace);    3
```

VSTUP: aktuální rotace objektu, u kterého chceme zjistit, jaký je vektor lokálního směru nahoru, vpřed a vlevo

1 2 3

VÝSTUP: vektor žádaného směru, např. pro použití s llTargetOmega

POZNÁMKA: funkce se používají na zajištění stejné rotace bez ohledu na natočení objektu

```
float llGetMass();
```

VSTUP: nic

VÝSTUP: vrací „hmotnost“ objektu v lindogramech, hodnotu, která se používá v PHYSICAL funkcích

POZNÁMKA:

- při volání z child primu vrátí hmotnost child primu, při volání z rootu vrátí celkovou hmotnost linksetu
- pokud se volá z attachmentu, vrátí hmotnost avatara

```
default
{
    state_entry()
    {
        llOwnerSay((string)llGetMass());
    }

}
```

`llSetForce(vector sila, integer lokalni);`

VSTUP:

- ***sila:*** vektor síly, tato síla bude působit na objekt kontinuálně
- ***lokalni:*** pokud je 1 (TRUE), bude vektor síly vztažen k lokálním souřadnicím objektu, pokud 0 (FALSE), vztahuje se k souřadnicím regionu

```
// Tento PHYSICAL objekt se bude vznaset
default
{
    state_entry()
    {
        float hmotnost = llGetMass();
        float grav = 9.81;
        llSetForce(hmotnost * <0.0,0.0, grav>, FALSE);
    }

}
```

```
vector llGetForce();
```

VSTUP: nic

VÝSTUP: vektor aktuální síly působící na objekt

```
vector llGetVel();
```

VSTUP: nic

VÝSTUP: vektor aktuálního směru a rychlosti pohybu objektu

llApplyImpulse(vector *sila*, integer *lokal*);

VSTUP:

- ***sila:*** vektor síly, tato síla bude působit na objekt jednorázově
- ***lokal:*** pokud je 1 (TRUE), bude vektor síly vztažen k lokálním souřadnicím objektu, pokud 0 (FALSE), vztahuje se k souřadnicím regionu

```
// Tento PHYSICAL objekt po kliknuti vyskoci
// do vzduchu a zase dopadne na zem
default
{
    touch_start(integer total_number)
    {
        vector sila = llGetMass() * <0.0, 0.0, 10.0>;
        llApplyImpulse(sila, FALSE);
    }
}
```

Příklad: Teleport pomocí Warp

- SL nemá funkci na skutečný přesun avatara na jiné místo, v rámci regionu lze však toto simulovat několika alternativami

- alternativa Warp pracuje na principu
    1) avatar se posadí na objekt
    2) objekt se přemístí na cílovou lokaci
    3) avatar automaticky opustí objekt
    4) objekt se vrátí na původní lokaci

Zdrojový kód 1/2: Teleport

```
vector CilTeleportu = <230.0, 100.0, 55.0>;
vector OriginalniPozice;

// funkce na presun objektu
Warp(vector cil)
{
    do
    {
        llSetPos(cil);
    } while ( llVecDist (llGetPos(), cil) > 0.01);
}
```

```
default
{

    state_entry()
    {
        // nastavi standardni akci pri kliknuti na sednuti
        llSetClickAction(CLICK_ACTION_SIT);
        llSetSitText("Teleport");
        // nastavi, ze si avatar sedne 0.5 metru nad objekt
        llSitTarget(<0.0,0.0,0.5>, ZERO_ROTATION);
        OriginalniPozice = llGetPos();
    }
```

Zdrojový kód 2/2: Teleport

```
    changed(integer change)
    {
        if(change & CHANGED_LINK)
        {
            // detekce avatara, ktery si sednul na objekt
            key Avatar = llAvatarOnSitTarget();
            if(Avatar != NULL_KEY)
            {
                Warp(CilTeleportu);
                llSleep(1);
                llUnSit(Avatar);
                llSleep(1);
                Warp(OriginalniPozice);
            }
        }
    }
}
```

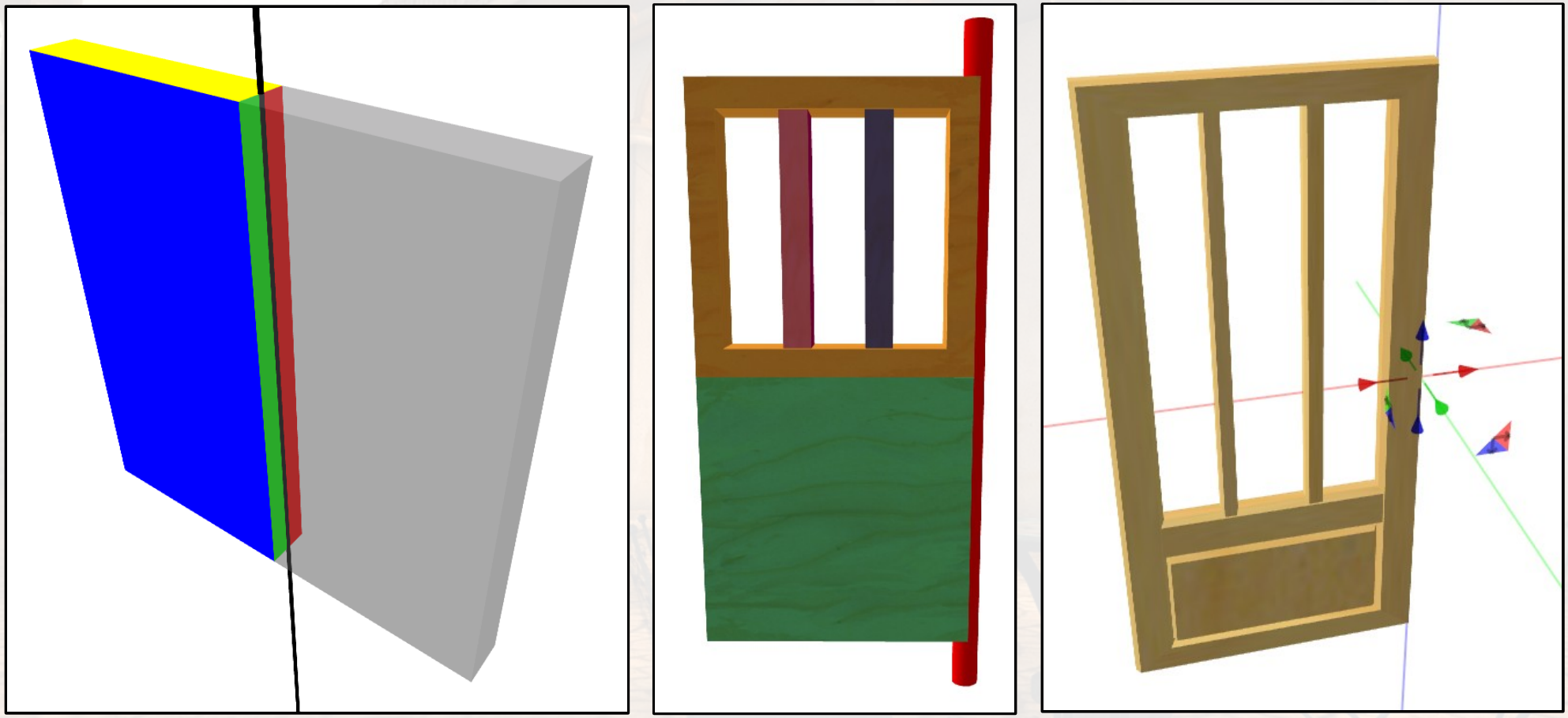
Příklad: Dveře

- Dveře detekují kliknutí a otevřou se, při dalším kliknutí nebo uplynutí určité doby se zavřou

- Dveře využívají rotace kolem osy Z root primu, jsou tedy tři základní metody, jak dveře vyrobit:

    - kvádr o rozměrech např. 3 x 0.1 x 3 metru a nastavit *Path cut* na B: 0.125, E: 0.625
    - libovolný linkset, root prim bude např. úzký válec na straně dveří
    - hotové sculpted dveře s osou na kraji

Příklad: Dveře



ilustrace - výroba dveří

Zdrojový kód 1/2: Dvere

```
// automaticky zavrit po X sekundach
float AutoZavreni = 15.0;
// o kolik stupnu otevrit dvere
float Uhel = 110.0;

dvere(integer Otevrit)
{
    vector RotovatRad;
    rotation Rotovat;

    if (Otevrit)
    {
        llSetPrimitiveParams([PRIM_PHANTOM, TRUE]);
        RotovatRad = <0, 0, Uhel * DEG_TO_RAD>;
        Rotovat = llEuler2Rot(RotovatRad);
        llSetRot( Rotovat * llGetRot() );
    } else {
        RotovatRad = <0, 0, -Uhel * DEG_TO_RAD>;
        Rotovat = llEuler2Rot(RotovatRad);
        llSetRot( Rotovat * llGetRot() );
        llSetPrimitiveParams([PRIM_PHANTOM, FALSE]);
    }
}
```

Zdrojový kód 2/2: Dvere

```
default
{
    touch_start(integer num)
    {
        dvere(TRUE);
        state open;
    }
}
```

```
state open
{
    state_entry()
    {
        llSetTimerEvent(AutoZavreni);
    }

    touch_start(integer num)
    {
        llSetTimerEvent(0.0);
        dvere(FALSE);
        state default;
    }

    timer()
    {
        llSetTimerEvent(0.0);
        dvere(FALSE);
        state default;
    }
}
```

Otázky a diskuze