TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky

a mezioborových studií

Studijní program: N 2612 ELEKTROTECHNIKA A INFORMATIKA

Studijní obor: Informační technologie

Interaktivní 3D-model sídla FM

Interactive 3D-model of FM residence

Diplomová práce

Autor: Lukáš Burda

Vedoucí DP práce: RNDr. Klára Císařová, Ph.D.

Konzultant: RNDr. Klára Císařová, Ph.D.

V Liberci dne: 6.5.2010

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování:

Na tomto místě bych rád poděkoval především vedoucí této diplomové práce RNDr. Kláře Císařové, Ph.D. za konzultace, poskytnuté materiály a plnou podporu v průběhu tvorby a také všem členům rodiny a přátelům, kteří aplikaci testovali a kritizovali.

Anotace

Tato práce zahrnuje rozsáhlý záběr tvorby interaktivní webové aplikace za použití nejmodernějších technologií a konceptů. Obsahuje množství teoretických i praktických postupů, znázorňujících propojení moderní trojrozměrné grafiky, nejpoužívanější webové technologie a stále víc se rozvíjející interaktivní obsah na technologii Adobe Flash.

Postupně jsou pojednány zásady a skutečnosti, které dávají umělým 3D objektům skoro dokonalou iluzi známou z reálných scenérií. Jsou použité technologie PHP, MySQL a s tím spjaté databáze, autodesk Maya pro tvorbu trojrozměrné grafiky, Adobe Dreamweaver jako aplikace pro práci s PHP kódem, Actionscript 3.0 v aplikaci Flash společně tvořící komplexní systém pro navigaci v budově A Fakulty mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci.

Annotation

This thesis contains wide range of creation of interactive web application with help of latest technologies and concepts. There are many theoretical and practical ways how to connect advanced CG graphics, most widly used internet technologies and fast developing interactive Autodesk Flash content.

This work elaborates ways and principals which give artificially created objects believable illusion of reality. Used technologies are PHP, Mysql and adherent databases, Autodesk Maya for 3d materials, Adobe Dreamweaver for PHP framework, Actionscript 3.0 within Flash application building all together a komplex system for navigation in building A of Faculty of Mechatronics, Informatics and Interdisciplinary Studies of Technical University of Liberec.

obsah

Úvod

Moderní technologie se pomalu stávají součástí životů každého člověka. Není to příliš daleko do minulosti, kdy první počítače zvládající základní výpočty začaly být dostupné pro pár jedinců, kteří si mohli tento "luxus" dovolit. Dnes má skoro každý k dispozici extrémně výkonné technologie jako jsou mobilní telefony, notebooky a podobně a rozšiřuje jejich používání i na dříve nemožné aktivity. Jednou z oblastí, kde je zaznamenám největší rozvoj, jsou právě počítačové technologie a s tím spojené používání internetových aplikací. Původní jednoduchá webová rozhraní jsou čím dál častěji nahrazována plně interaktivními prostředími. Čtverečkové obrázky jsou pozvednuty na hyper realistické trojrozměrné zpracování a právě do tohoto segmentu směřuje moje práce.

V této publikaci se pokusím vnést vhled do problematiky nejmodernější trojrozměrné grafiky a ukáži jaké jsou procesy tvorby umělé reality, kde člověk není schopen říct rozdíl mezi fikcí a skutečností a tyto principy pomalu přenesu a propojím s moderními Flash aplikacemi. Samotný Flash vidíme skoro na každé stránce na internetu a i když se zdá být jednoduchý, tak i zde je s novými verzemi mnoho úskalí, které programátor a grafik musí překonat. Právě implementace 3D grafiky do prostředí Flash je jedno z těch lákadel, která mě vedou k této práci a také rozšíření o dynamický grafický obsah generovaný z programů pro tvorbu pokročilého 3D prostředí zaimplementovaného do obsahu snadno aktualizovaných databází.

1 Moderní metody 3D

1.1 Úvod do dnešního 3D

Metody a programy, které jsou dnešním standardem pro zpracování počítačové grafiky, se vyvíjí již téměř dvě desetiletí a produkty jimi vytvořené jsou blíže realitě než kdy jindy. Rozdíly mezi skutečným obrazem a tím, co je vytvořené pomocí počítačové grafiky jsou pomalu ale jistě smazávány a otázka, zdali je to, na co se díváme, vytvořené uměle nebo ne, je stále častější. Dokonce i lidé, zabývající se CG (cyber graphics) počítačovou grafikou, mají mnohdy rozporné názory, avšak stále se vyskytují drobné indicie, které odlišují běžnou realitu, vnímanou lidským okem od té umělé.

Největší rozdíl, který přemosťuje dvacet let vývoje grafických balíků, není v postupu tvorby modelů, podkladů nebo získávání referencí a návrhů, ale v té poslední fázi tvorby, která se nazývá rendering a postprodukce.

1.2 Rendering a Render farmy

K realizaci maximálně realistického CG je potřeba také obrovský výpočetní výkon. S rostoucí kvalitou grafiky je potřeba i úměrně rostoucího výpočetního výkonu.

Výstup z některého 3D programu je vypočítáván procesorovou jednotkou nebo, pokud se jedná o realtimové renderery, tak grafickým jádrem. Tento výstup zahrnuje nejnovější technologie jako je reaytracing, displacement, ambientoclusion, depth of field a podobně. Jako malý demotivující příklad náročnosti na výpočetní výkon může bý třeba

film Transformers. Zde jeden snímek (jedna vteřina filmu jich má 24) zabral "renderovací farmě" o počtu několika tisíc procesorů klidně i 24 hodin, ale výsledek je naprosto unikátní.

1.3 3D packages

Pro různé účely jsou k dispozici 3D programy počínaje architektonickými až po designerské. S rozvojem počítačové grafiky se objevují stále nové programy na její zpracování. V tomto projektu je používám software Autodesk Maya, který se řadí mezi profesionální 3D nástroje s rozšířenými možnostmi modelování, materiálů, renderingu a animace. Verzí tohoto programu je na trhu mnoho, ale hlavní výhody přináší nejnovější verze. Samozřejmou implementací v nových verzích programu Maya je i profesionální renderovací zásuvný modul Mental Ray, který podporuje všechny moderní technologie renderingu.

1.4 Mental Ray realistic materials

1.4.1 Standardní materiály v Maye vs Mental ray materiály

Podmínkou pro vytváření realistického 3D jsou materiály. Každý vývojový balík má svou vlastní sadu standardních materiálů pro všeobecné použití. V prostředí Autodesk Maya se jedná o souhrn asi deseti materiálů, z nichž má každý své specifické použití. Materiál Lambert je vhodný pro simulaci matných objektů jako jsou stěny, kameny, a jiné nelesklé objekty. Pak je k dispozici také blinn materiál, který se hodí pro simulaci kovů a lesklých předmětů a v neposlední řadě phong materiál, který exceluje v simulaci plastů s velkou odrazivostí.

Všechny tyto materiály jsou spolehlivé a vyskytují se už od prvních verzí programu Maya. Renderer, který byl navržen pro program maya s podporou těchto standardních materiálů, je Maya Software Renderer. Výsledky z tohoto enginu jsou kvalitní, ale ne zcela vyhovující.

Od verze Maya 6 je možné implementovat do prostředí Mental Ray Renderer, který byl ve své době průlomovým pluginem pro rendering fotorealistických obrázků. S ním přichází i fyzikálně přesné materiály pro použití s nevyššími nároky na kvalitu výsledků. Jsou to car paint shadery pro přesné simulace nejen automobilových laků, DSG materiály pro přesné fyzikální simulace chromatických materiálů, dielectric materiál fungující jako kvalitní podklad pro tvorbu vody, gelů a jiných průhledných materiálů, sss_materiály k simulaci organických hmot a objektů rozptylujících a pohlcujících světlo a tímto výčtem tento přehled nekončí.

Podle mého názoru je největším průlomem v oblasti shadingu s použitím mental ray rendereru nový materiál (přicházející s verzí Maya 2008) s označením mia_material a mia_material_x.

Jde o skloubení mnoha různých materiálů do jednoho intuitivního a ohebného celku. Materiál je fyzikálně přesný, využívající fyzikálních poznatků pro správnou definici výsledného vzhledu objektu materiálem. V zásadě může tento materiál simulovat většinu pevných povrchů, průhledných povrchů a naráží pouze u aplikace na organické materiály. Zde je lepší použít sss_materiály.

Fyzikální fungování je zajištěno kopírováním realistického fungování materiálů v přírodě. Vzhled každého reálného materiálu je výsledkem tří hlavních veličin. Diffuse color je vlastní barva materiálu a říká, kolik světla materiál odráží nebo pohlcuje. Druhá veličina je reflectivity (odrazivost). Znamená, jak moc je výsledný vzhled ovlivňován okolními odrazy, tedy kolik světla se odráží a tvoří odrazy. Třetí hlavní veličina je transparency (průhlednost). Je-li objekt průhledný, tak neodráží tolik světla, ani ho nepohlcuje, ale světlo prochází skrze objekt.

Všechny tyto základní veličiny mají hodnotu od nuly do jedné. Výsledný součet pak musí být menší než jedna vlivem zákona zachování energie a ztráty energie vlivem dalších veličin. Pokud se zachovají tyto podmínky, tak výsledný vzhled bude blízký realitě a použitím pokročilých renderovacích technik bude výsledek fyzikálně přesný.

1.5 Metody, které přibližují CG realitě

Pro vytvoření realistického CG (cyber graphics) je potřeba zpracovávat projekt s ohledem na tento cíl. Umělec nemůže spoléhat na to, že při renderingu bude mít realisticky vyhlížející obraz, pokud nebude pracovat systematicky na tomto výsledku. Je třeba mít na paměti, že pokud se nedotáhne jedna část vývoje projektu, pak výsledek nebude mít kýženou hodnotu.

Kvalitně zpracované modely, textury, osvětlení a podobně jsou pouhou předehrou pro následující krok. Rendering je dnes nejdůležitější proces, kde lze z jednoduchého, však správně připraveného, podkladu vytvořit naprosto dokonalé fotorealistické výsledky. Následující techniky tuto realističnost poskytují.

1.6 Ray Tracing

Raytracing je metoda jak počítat cestu vlnových částic skrze různé systémy, ať už se jedná o vzduch, vakuum nebo vodu. Této metody se využívá ve dvou moderních odvětvích. První je fyzikální ray tracing, kde se výpočtem získávají data o optických a jiných systémech a druhé využití je v dnešní počítačové grafice. A právě toto odvětví nás bude zajímat.

Metody generování obrázků v počítačové grafice jsou různé s různou kvalitou výstupu, ale právě raytracing je metoda fyzikálně nejpřesnější. Spočívá v generování výsledného obrázku trasováním, neboli sledováním cesty paprsku světla, skrze pixely na ploše obrázku. Fyzikální přesnost tohoto výpočtu zajišťuje vysokou realističnost výsledného

obrázku většinou mnohem větší než u tradičních renderovacích technik, ale za cenu vysokého renderovacího času a potřeby výkonnější výpočetní síly.

Tyto klady a zápory dělají z raytracingu metodu vhodnou pro generování obrázků, kde není potřeba realtimového renderování a lze zde počítat obrázky dopředu po delší dobu. Patří sem například filmový průmysl, televizní speciální efekty a podobně. Naopak se nehodí pro počítačové hry, kde je potřeba renderovat velké množství obrázků za co nejnižší čas. Je nutné podotknout, že současně jsou již vyvíjeny metody, jak používat raytracing i v herním průmyslu. Jedná se o verze značně omezené, ale dodávající herním prostředím větší realističnost.

Raytracing dokáže simulovat velké množství efektů, jako jsou odlesky (reflections), lomy světla (refraction) a rozptyl světla (scattering). Ray tracing je také základem pro globální iluminaci a kaustiku.

1.6.1 Ray Tracing algoritmus obecně

Trasování optického paprsku představuje realističtější metodu generování obrázku ve 3D prostředí než ray casting (viz. Ray casting) nebo scanline rendering (viz. Scanline rendering). Pracuje na principu trasování paprsku z imaginárního oka pozorovatele skrze každý pixel obrázku a počítání barvy každého objektu, kterým tento paprsek prostupuje. Scéna je pro raytracing zpracována podle programátora nebo pomocí dodatečných nástrojů 3D designera. Barvu objektu ve finále ovlivňují i další jevy, jako jsou barvy textur, modelů a prostředí, stejně jako je tomu i u běžného fotografování.

Normálně musí být každý paprsek testován na průnik s objekty ve scéně. Po identifikování nejbližšího objektu je odhadnuto příchozí světlo a bod průniku, materiál objektu, jeho barva a poté je vypočítána výsledná barva daného pixelu. Osvětlení daného pixelu je pak dáno také množstvím nepřerušených paprsků, které na pixel dopadají. Některé algoritmy pro osvětlení, odrazy a lomy potřebují několikanásobné trasování paprsku ve scéně pro realistické zobrazení.

V podstatě jsou dvě metody jak sledovat paprsky, a to do kamery a od kamery. V reálném světě se jedná o sledování paprsků do kamery jak je nám jasné. Obraz, který vnímá naše oko je ale vytvořen pouze minimálním množstvím paprsků světla. Pokud bychom stejný princip chtěli použít i ve 3D scéně, tak bychom museli počítat obrovské množství paprsků, z nichž většina ani nezavadí o kameru, ze které se díváme, tudíž bychom zbytečně plýtvali výpočetním výkonem. Proto se v 3D grafice používá většinou druhý způsob, a to trasování paprsků od kamery do scény. Takto se využijí všechny paprsky, které vysíláme, protože vždy budou v oblasti imaginárního obrazu a můžeme si dovolit větší hloubku trasování bez plýtvání výpočetní silou.

Metoda, která trasuje paprsky od zdroje světla do kamery je nazývána Photon Mapping (mapování photonů) a je výpočetně náročnější než raytracing, ale dává ještě lepší výsledky.

1.6.2 Co se děje s paprskem v realitě

V přírodě je paprsek emitován ze světelného zdroje. Ten pak putuje prostorem dokud nakonec nenarazí do blokujícího objektu. Tento paprsek lze chápat jako proud světelných částic (fotonů), které putují po stejné dráze. Ve vakuu by tato dráha byla dokonale přímá (pokud ignorujeme teorii relativity), ale v realitě mohou nastat různé kombinace čtyř jevů: absorbce, odrazu, lomu a fluorescence.

Povrch objektu může odrážet všechny složky světla do různých směrů s následným snížením intenzity odražených nebo lomených paprsků. Průhledné materiály zase lomí některé složky světla do sebe a některé se pohlcují s možností následného ovlivnění barvy lomeného paprsku. V některých případech může povrch znovu fluorescentně emitovat světlo v náhodném směru v dlouhých vlnových délkách.

Mezi odrazem, lomem a fluorescencí platí zákon zachování energie. V praxi to tedy znamená, že pokud objekt odráží 60% světla, tak již nemůže lomit dalších 50% světla, protože by výsledná hodnota původní energie byla 110%. Tohoto principu se využívá ve

fyzikálně přesných materiálech, kde výsledná hodnota je daná právě závislostí odrazivosti, průhlednosti a absorbce paprsků světla. Tyto vlastnosti se nachází v materiálech pokročilých renderovacích enginů jako je Mental Ray nebo V-Ray.

1.6.3 Průlom v Ray Tracingu

Autor velice důležitého průlomu ve výpočtu Ray Tracingu je Turner Whitted. Od jeho jména pochází Turner Whitted Ray Tracing. V roce 1979 rozšířil původní algoritmus, který trasoval paprsky z oka pozorovatele pouze ke kolizím objektů a zde končil, o další fáze, které mohly generovat tři typy paprsků: odraz, lom a nebo stín. Podle zákonu odrazu a lomu z optiky pak paprsek pokračuje ve své cestě. První objekt, na který narazí odražený paprsek, je pak viditelný jako odraz. Stejně to funguje i pro lomený paprsek. Paprsek stínu pak testuje, zdali je objekt, na který dopadá, viditelný pro světlo a zabraňuje tím dalšímu trasování paprsku na tento objekt. Prakticky to funguje tak, že paprsek narazí do objektu v nějakém bodě. Pokud tento bod je viditelný pro světlo, pak je vypočítán paprsek mezi světlem a tímto bodem. Pokud v této cestě nachází nějaký neprůhledný objekt, pak je tento bod ve stínu.

Tento nový způsob počítání Ray Tracingu přispěl k větší realističnosti výsledných obrázků.

1.7 Ray Casting algoritmus

Vrhání paprsků (Ray Casting) je testování průniku paprsku a povrchu z pohledu oka nebo kamery. Tento pojem byl prvně použit v počítačové grafice v roce 1982 v článku autora Scott Roth popisujícím metody renderování CSG modelů.

CSG modely jsou modely sestavované z pevných těles, jako jsou kostky a válce za použití booleovských metod kombinování objektů. Často CSG reprezentuje modely, které vypadají značně složitě, ale jsou ve skutečnosti chytře skombinovaná pevná tělesa

Vrámci Ray Castingu se řeší:

Všeobecný problém určování pořadí objektů ve scéně.

Technika odstraňování schovaných objektů ve scéně založené na určování prvního protnutí paprsku generovaného z oka skrze každý pixel obrazu.

Nerekurzivní varianta ray tracingu, který vrhá pouze první paprsky.

Přímá metoda renderování objemů.

Ray Casting není synonymum pro Ray Tracing, ale lze ho považovat za přemostění a značně rychlejší verzi Ray Tracingu. Hlavní zjednodušení tohoto algoritmu spočívá v renderování pouze primárních cest paprsků a přerušení výpočtu po dosažení objektu, s kterým má průnik. Toto eliminuje možnost realistického počítání odrazů, lomů a stínů. Na druhou stranu všechny tyto elementy lze zfalšovat chytrým užitím textur se zakreslenými stíny, odlesky a podobně.

1.7.1 Scanline rendering

Scanline rendering je algoritmus pro rozlišování viditelných povrchů ve 3D grafice. Pracuje na bázi řádku za řádkem (row-by-row basis) místo polygon-by-polygon nebo pixel-by-pixel. Všechny polygony jsou nejprve seřazeny podle polohy v Y, kde se poprvé vyskytnou, po té je každý řáded počítáa za pomocí průniku skenované linie a polygonů na začátku seznamu. Během toho jsou ze seznamu objektů odstraňovány již dále neviditelné objekty, jak aktivní scanovací linie dál pokračuje obrázkem směrem dolů.

Za použití této metody není nutné přesouvat z hlavní paměti koordináty všech vertexů do pracovní paměti. Přesouváme pouze vertexy definující hrany, které protínají aktuální skenovanou linii a každý tento vertex je přečten pouze jednou. Vzhledem k tomu, že hlavní paměť je pomalá ve srovnání s cache, je nutnost malého přesunu dat z hlavní paměti velkým zrychlením.

1.8 Normal mapping a diplacement

1.8.1 Low polygons a high polygons

Polygonové objekty můžeme rozdělit podle členitosti na objekty s nízkým polygonovým počtem (lowpoly) a na objekty s vysokým polygonovým počtem (highpoly). Oba dva mají své klady a zápory.

S větším počtem polygonů je možné zobrazit větší množství detailů. Každý polygonový objekt je sestaven z plošek běžně tvořených třemi nebo čtyřmi vrcholy. Vezměme si nyní situaci, kdy je objektem lidský obličej. Plochu čela tvoří například dva polygony, tedy 6 bodů. Samozřejmě je jasné, že ze šesti bodů nelze vymodelovat detaily jako například vrásky kůže. Nevýhodou malého počtu polygonů je tedy málo detailů. Na druhou stranu je vyrenderovaní této plochy velice rychlé.

Přesně opačně je to u objektů s vysokým polygonovým počtem. Detaily jsou dobře viditelné, ale zvedá se renderovací čas. Stejně jako u Ray Tracingu mají i modely, podle počtu polygonů, různé uplatnění.

Existují dvě metody jak využít výhod lowpoly modelů a přitom překonat jejich nevýhody. Jedná se o displacement mapování a normal mapování.

1.8.2 Normal mapovaní

V počítačové grafice je normal mapping metoda jak vytvářet na jednoduchých modelech falešný dojem detailnosti, aniž bychom přidávali další polygony. Normal mapa je většinou barevná textura v RGB (red, green, blue) spektru, kde každé barevné spektrum koresponduje k X,Y a Z koordinátě normálových vektorů detailnějšího modelu, než je ten

mapovaný. Běžný postup je generování normálové mapy na lowpoly model z modelu o mnohem vyšším rozlišení.

1.9 Normálový vektor

Každá ploška (polygon) objektu ve 3D je dána třemi a více body. Kolmý vektor na tuto plochu se nazývá normálový vektor. Tento vektor je důležitý k výpočtu Lambertského (rozptýleného) osvětlení povrchu. Znamená to, že normálový vektor, který je rovnoběžný s paprskem od světelného zdroje, určuje polygon, který získá plné osvětlení (nehledíme-li na vlastnosti materiálu nebo jiné objekty ve scéně).

1.10 Jak funguje normal mapping

Představme si model koule. Tato koule má určitý počet polygonů a tedy i normálových vektorů. Na základě těchto vektorů získává model rozptýlené osvětlení. Pomocí normálového mapování však můžeme skrze RGB texturu zvětšit počet těchto vektorů na mnohanásobek. Každý pixel této textury má v sobě zakódovaný RGB barevný kanál. Barva tohoto kanálu koresponduje s prostorovými souřadnicemi x,y,z. Tyto prostorové atributy jsou relativní ke konstantnímu systému object-space normal mapy a nebo k prostoru odpovídajícímu koordinátám textury v UV rozložení nazývané tangent-space normal map. Získáme tak mnohem více detailů na povrchu modelu a to především v kombinaci s pokročilými osvětlovacími technikami.

1.10.1 Normal mapping v počítačových hrách

Interaktivní normálové mapování bylo původně pouze možné na PixelFlow, paralelním renderovacím stroji postaveném na University of North Carolina. Později bylo možně provádět renderování normálových map na high-end pracovních stanicích používajících multi-pass rendering a framebuffer operace (jedná se o výstupní zařízení, které

řídí video display z dat uložených kompletně v paměti) a nebo na low-end hardwaru se zpracováním paletových textur.

Od roku 2003 za se normal mapping hojně rozrostl díky používání shaderů a rozvoji herních konzolí. Popularita normálového mapování spočívá v nízké náročnosti na výpočet oproti jiným metodám. Velká část této nenáročnosti je daná implementací distance-index detail scaling. Znamená to, že s rostoucí vzdáleností objektu, potřebuje povrch méně detailů a ne tak komplexní světelné simulace.

Základní normal mapping může být implementován v jakémkoli hardwaru, který podporuje paletové textury. První herní konzole s podporou normálového mapování byla Sega Dreamcast. Nicméně Microsoft X-box byla první konzole s širokým užitím v běžných hrách. Pro zajímavost pouze Playstation 2 neměla podporu normal mappingu. Dnešní konzole a počítačový hardware již plně spoléhají na normálové mapování.

1.11 Displacement mapping

Jedna z alternativních technik v počítačové grafice jako je bump mapping, normal mapping a parallax mapping. Stejně jako zmíněné ostatní je displacement mapping používán k vytvářené hlubších detailů na jednoduchých polygonových modelech.

Používá se zde normálních nebo procedurálních textur a výškových map k vytvoření efektu, kde dochází k geometrickému posunu bodů objektu nad texturou aplikovanou na povrch obvykle podle normálového vektoru polygonů. Hodnota posunu bodů je daná hodnotou textury v daném místě. Dává to povrchu větší dojem hloubky a detailu, posilující vlastní zastínění, okluzi a měnící siluetu v renderovacím čase. Na druhou stranu se jedná o velice náročnou metodu kvůli rapidnímu zvětšování počtu bodů povrchu.

Porovnáme-li diplacement mapping a bump mapping, tak hlavními rozdíly jsou: Bump mapping vytváří dojem většího detailu na bázi pixelů a renderovacím čase a nepřidává rozlišení povrchu, na který je mapa aplikovaná. U displacementu je objekt v závislosti na mapě rozčleněn a nově vzniklé body jsou následně posunuty podle hodnoty

barvy textury a dochází tak ke změně siluety. Je proto vhodné některé displacement mapy vytvářet z detailnějších verzí objektu, na který se bude pak mapa aplikovat.

Po mnoho let byl displacement mapping specialita high-end renderovacích systémů jako PhotoRealistic RenderMan, zatímco realtimové API jako OpenGL a DirectX tuto možnost neměli. Jeden z důvodů proč tomu tak bylo, byla absence originální implementace adaptive tesselation (přizpůsobivého rozčlenění) povrchu, který by dosáhl takového detailu rozčlenění, aby bylo možné dosáhnout rozlišení pixelu příslušného výstupního zobrazení.

1.11.1 Výhody DM

Displacement mapping šetří paměť grafické karty a paměťovou sběrnici. Vrcholy, které tvoří jednotlivé trojúhelníky, jsou simulovány pomocí dvourozměrných textur obsahujících všechny potřebné informace a není nutné mít tyto vrcholy fyzicky na objektu.

Displacement mapping dokáže mnohdy vytvořit mnohem detailnější výsledek než by bylo možné samotným modelováním detailů na objektu.

Výhody displacement mappingu se projeví u animování objektů, kde není nutné pracovat s tisíci body, ale pouze s malým množstvím.

Jelikož jsou všechny informace o objektu převedeny do 2D textur, lze na ně aplikovat veškeré poznatky, které máme se zpracování grafiky jako je komprese, prolínání, efekty a podobné.

1.11.2 Nevýhody DM

Displacement mapy nejsou použitelné na všechny typy modelů. Modely s velkým počtem úhlových zlomů, modely hladké nebo mající na sobě různě směrované plochy jako u krystalů, jsou mapy nepoužitelné, protože deformují daný reliéf.

Displacement mapping upravuje povrch pouze jedním směrem, většinou lokálním normálovým, tudíž není použitelný na vysoce členité povrchy, kde je potřeba upravovat povrch v různých směrech. V těchto případech je nutné detaily modelovat ručně.

1.12 HDRI

Při zpracování obrazu, v počítačové grafice a fotografii, je High Dynamic Range obraz (High Dynamic Range Image - HDRI) technika umožňující rozsáhlejší dynamický rozsah expozice mezi nejsvětlejším a nejtmavším místem obrazu, než je možné u běžného digitálního zpracování obrazu nebo u fotografických metod. Tento širší rozsah umožňuje HDR obrázkům zachytit přesně široké spektrum osvětlení v reálných scenériích v rozmezí od slunečního světla ke světlu nočnímu.

Dva hlavní zdroje HDR obrazů jsou počítačový rendering a skládání vícero fotografií dohromady. Takto vznikají takzvané low dynamic range LDR fotografie a obrazy. Následným použitím metody mapování barevných tónů lze zredukovat nebo naopak vysaturovat oblasti pro realističtější rozsah nebo umělecké efekty.

V oblasti fotografie je dynamický rozsah měřen v EV rozdílech (Exposure Value - hodnota expozice, známých také jako stops) mezi nejsvětlejším místem a nejtmavším místem. Nárůst EV o jeden stop je roven navýšení světla o dvojnásobek. Je to tedy reprezentováno jako logaritmus o základu 2. EV difference = log2(contrast ratio).

High dynamic range fotografie jsou většinou tvořené skládáním (prolínáním) většího počtu fotografií. Některé fotoaparáty mají již v základu zabudovanou možnost zachycení HDR.

1.12.1 HDR v počítačové grafice

HDRI v počítačové grafice znamená nasvětlení 3D scény bez použití světel, ale pouze za použití speciálního HDR obrazu. HDR obraz obsahuje až 32 bitovou informaci

o veškerých zdrojích světla v celé scéně. Není potřeba tedy používat standardní světla a redukuje se tak i renderovací čas. Tato metoda osvětlení je velice vhodná při renderingu lesklých objektů, kterým dodává vysoký stupeň realističnosti a také se používá při kompozici 3D objektů do reálných scenérií, protože osvětlení metodou HDR vypadá velice realisticky.

Běžně bitmapy pracují s 8-mi bitovou barevnou hloubkou pro každou jednu barvu, to znamená rozsah 0-255. HDR má rozsah až 32 bitů, to činí rozsah prakticky neomezený. Barevná informace je uložena v číslech s plovoucí řádovou čárkou např.: 3567655,87657. Možnosti jsou tedy veliké. Ve formátu HDR je uložena hodnota světla pro každý pixel, tedy je zde již zakomponované, jak bude předmět vypadat.

1.12.2 HDR ve 3D aplikaci

Každý 3D package má svůj vlastní způsob jako vytvořit high dynamic range osvětlení. V prostředí Maya je většinou HDR obraz aplikován jako kulová projekce na kouli. Podmínkou použití HDR při renderování je renderovací engine, který umí s tímto formátem pracovat. V případě Autodek Maya je to implementace Mental Ray rendereru.

Nejvíce viditelné změny při použití HDR osvětlení jsou realistické odlesky a jemné stíny podle high dynamic range obrazu, který použijeme. Někdy je také potřeba HDR mapy upravit v některém z 2D grafických programů a nastavit ji degamma vlastnost na přijatelnou hodnotu.

1.13 Ambient occlusion

Ambient occlusion je metoda v počítačové grafice, která pomáhá dodat realismus modelu vzetím v úvahu zeslabení světla vlivem zastínění okolím. AO se snaží přiblížit způsob, jakým se chová světlo v reálném světě a to především u nereflektivních objektů.

Narozdíl od phongovského stínování, je ambient oclussion globální funkce, kde osvětlení každého bodu je funkce obklopujících modelů ve scéně. Způsob jak je simulováno osvětlení s pomocí ambient oclussion je srovnatelný s tím, jak je objekt osvětlen za zataženého dne.

Ambient occlusion je většinou počítané generováním paprsků ve všech směrech od objektu. Paprsky, které dosáhnou pozadí nebo oblohy zvětší světlost povrchu a paprsky, které narazí na nějaký další objekt přispívají k menšímu osvětlení. Ve výsledku je objekt, který je obklopen množstvím jiných objektů, renderován jako tmavý a naopak. Ambient oclussion nefunguje pouze mezi různými objekty, ale také na jeden objekt, kde různé záhyby fungují jako blokující objekty. Lze takto získat mnohem prokreslenější strukturu objektu.

Ambient occlusion je poměrně populární metoda ve 3D grafice vzhledem k její jednoduchosti a relativní výpočetní nenáročnosti. V produkční oblasti je označován také jako „sky light".

1.13.1 Ambient occlusion matematicky

Okluze A_p v bodě p na povrchu s normálou N lze vypočítat jako integrál funkce viditelnosti (visibility function) nad hemisférou Ω s ohledem na úhel primitiva:

$$A_p = \frac{1}{\pi} \int_{\Omega} V_{p,\omega} \, (N \cdot \omega) d\omega$$

Kde $V_{p,\omega}$ je funkce viditelnosti p, definované jako nula, pokud p je v okluzi ve směru ω a naopak. Jsou používány různé techniky jak aproximovat tento integrál v praxi. Nejspíše nejvíce jednoznačná technika je použití Monte Carlo metody pouštěním paprsků z bodu p a hledání průniků s okolními objekty.

Jako doplněk k hodnotě ambient occlusion je často generována normála ohnutí. Tento vektor ukazuje ve středním úhlu vzorků bez okluze. Tento vektor lze použít pro aproximaci HDR ve scéně s AO.

1.14 Depth of field (hloubka ostrosti)

Pro ještě větší realističnost a přiblížení výsledného renderu ve 3D realitě se používá algoritmus pro výpočet hloubky ostrosti neboli DOF (depth of field).

V optice, především ve fotografii a filmu a neposledně počítačové grafice, je hloubka ostrosti DOF ta část scény, která se zobrazuje v přijatelné ostrosti. Podobně i čočka může dobře zaostřit pouze na jednu vzdálenost. Pokles ostrosti je postupný, od stran ostřící vzdálenosti, takže DOF je ostrá oblast daná ostřící vzdáleností a poloměrem ostrosti.

V závislosti na dané situaci je někdy potřeba mít zaostřenou kompletní scénu a tím pádem mít velkou hloubku ostrosti, a naopak někdy je vyžadován malý ostrý úsek s použitím malé hloubky ostrosti. Díky možnosti měnit DOF v reálném čase, lze dosáhnout opravdu zajímavých efektů, které mohou odlišovat dobré 3D zobrazení od fotorealistického. Velká hodnota DOF se také označuje jako deep focus a malá hodnota se označuje jako shallow focus.

DOF je určeno vzdáleností subjektu, tedy vzdáleností plochy která je perfektně zaostřená, dále ohniskovou vzdáleností čočky, clonovým číslem (f-number), velikostí obrazového snímače a rozptylovým kruhem.

V počítačové grafice se DOF většinou počítá podle pravidel, která platí i ve fotografování a filmu. Výpočet se provádí jednoduchými algoritmy až po fyzikálně přesné výpočty s ohledem na mnoho reálných veličin. Běžné renderovací enginy podporují jednoduché nastavení hloubky ostrosti s poměrně vysokým nárokem na výpočetní výkon a proto se vyplatí pro renderování používat některé fyzikálně přesně renderovací enginy jako je Mental Ray for Maya. Zde může operátor vkládat do kamery připravené fyzikální „čočky" (lens shaders).

1.14.1 Matematický princip DOF

Pro danou velikost snímacího čipu na střední vzdálenost, je hloubka ostrosti přibližně určena zvětšením subjektu a clonovým číslem. Pro dané clonové číslo se zvětšujícím zvětšením, buď pohybem kamery k objektu nebo zvětšováním ohniskové vzdálenosti, se zmenšuje rozostření a naopak.

1.15 Motion blur

Motion blur je velmi často viditelný jev ve fotografii, videu a s rostoucí měrou i v počítačové grafice. Při fotografování vytváříme obraz, který nereprezentuje pouze jediný okamžik v čase, ale kvůli technologickému omezení zaznamenává několik událostí najednou. Technologie je schopna zaznamenat data za určitou časovou jednotku a proto výsledný obraz reprezentuje sled několika poloh objetku, který se snažíme zachytit. Rozmazaný obraz je pak výsledkem složení všech zaznamenaných poloh objektu.

V počítačové grafice jsou dvě hlavní metody jak simulovat motion blur. Jedna metoda je zachytit pravý motion blur aproximací všech poloh objektu. V 3D animaci představuje každý snímek perfektní instanci času s nulovým rozostřením. Při počítání pravého motion bluru je ke každému snímku přidáno několik snímků z obou stran časové osy a následně je vypočítána vzdálenost a podle toho rozostřen výsledný render. Tato metoda je časově a výpočetně velmi náročná a tak se využívá také nepravý motion blur.

Nepravý motion blur je počítán bez přidávání mezi snímků a bere se v potaz pouze předcházející snímek a podle toho se rozostřuje výsledný obraz. Toto není tolik náročné na výpočetní výkon, ale výsledky nejsou tolik přesné.

1.16 Global illumination

Globální iluminace (GI) je název pro sadu algoritmů používaných ve 3D počítačové grafice k dosažení fotorealistického osvětlení scény. Tyto algoritmy berou, kromě přímého osvětlení světelnými zdroji (direct illumination), v potaz také osvětlení vznikající odrazy světelných paprsků (indirect illumination) od objektů ve scéně, ať už jsou reflektivní nebo ne.

V počítačové grafice je globální iluminace souhrn rozptýleného odrazeného světla a kaustiky (global illumination and castics). Všechny objekty, které se nacházejí ve scéně pak ovlivňují výsledné osvětlení scény a vytvářejí mnohem realističtější osvětlení.

Výsledky renderingu s GI jsou realističtější, ale to za cenu větší výpočetní náročnosti a následující pomalejší generování renderu. V případě renderování mnoha snímkové animace je pak výpočetní a časová náročnost obrovská. K tomu ještě s opakovaným přepočítáváním GI dochází k tvorbě artefaktů vlivem změn v rozložení iluminačních bodů. Moderní renderovací enginy ovšem přicházejí s možností uložení GI mapy do externích souborů a tím je možné minimalizovat renderovací časy a zamezit tvorbě artefaktů.

Algoritmus pro rozptýlené vnitřní odrazy (diffuse inter-reflection), který zajišťuje výpočet odrazů a vlastností světla při dopadu na nerovný povrch, zajišťuje potřebné rozptýlení světla do prostředí a spojuje schopnost výpočtu odlesků a jejich odrazů.

GI ve 3D software je počítáno na principu generace photonů ze světelných zdrojů. Každý foton je vystřelen ve směru světelného zdroje s nastavenou energií. Tato energie říká, jako silou se bude částice pohybovat prostředím, a tedy jak daleko dorazí a při kolizi s nějakým objektem také určuje, jaké zanechá v daném bodě osvětlení a po odrazu jak daleko bude putovat a tak dále. V softwarovém balíku se dá ovlivnit limit pro počet odrazů photonu, jeho energie, barva, kterou nese a zanechá při kolizi, poloměr osvětleného kruhu v místě dopadu a další parametry.

Při emitování photonů je nutné také zajistit potřebné množství částic pro danou scénu. Platí pravidlo, že čím větší je scéna, tím více musí být fotonů pro GI, ať už zvětšením počtu emitovaných částic na světle nebo zvětšením počtu emitujících světel.

Pokud chceme mít efekty photonů procházejících průhlednými objekty, pak je nutné povolit také caustics, které stejně jako GI pracuje na principu emitovaných fotonů a vytváří dobře známé skleněné či vodní efekty rozptýlením nebo koncentrováním částic.

2 3D v aplikaci „Interaktivní 3D navigace v budově A“

2.1 Panorama haly budovy A

Použitím výše zmíněných technik jsem vytvořil podklad pro základní panorama v budově A. V následujících řádcích se pokusím shrnout základní principy, které jsem aplikoval při tvorbě nejen haly, ale i kompletních interiérů v průletech různými patry.

2.1.1 Polygon to subdivision modelování

Profesionálními vývojáři nejčastěji používaná technika vytváření modelů je polygon to subdivision modelování. Hodí se na vytváření pevných povrchů, ostrých hran zdí a definování dalších tvarů potřebných v architektonické vizualizaci. Vizualizace mohou být vytvářeny pro různé situace a tak se hodí různé postupy a dosahuje se rozdílných detailů pro části scény. Jeden z populárních postupů je tvorba scény pro kameru. Je dána pevná kamera a její rozsah při případném pohybu a modely jsou vytvářeny v tomto vztahu. Umělec si tak může dovolit omezovat množství detailů tam, kde nebudou při pohybu vidět.

Pro potřeby modelu haly budovy A tento přístup není vhodný kvůli kompletní volnosti kamery v interiéru. Detaily tedy musely být konzistentní ve všech částech modelu a o to pracnější a časově náročnější je tvorba této scény. Model budovy A je založen na přesných fotografických podkladech s architektonickými půdorysy. Proto jsou modely proporčně přesné a simulují tak lépe prostory budovy. Návštěvník aplikace si pak snadno odvodí kde se nachází.

Modely jsou tedy vytvářeny z jednoduchých polygonů s následným vyhlazováním převodem na sub-division povrchy a dodáváním potřebných detailů. Některé detaily je samozřejmě lepší dotvořit v renderingu texturami. Postup tvorby vychází z klasického sochařského postupu od jednoduchých tvarů k detailům.

2.1.2 UV mapping, textury a procedurální textury

Modely jsou po vytvoření připraveny na texturování. Textury se aplikují v zásadě dvěma způsoby. Projektováním a aplikováním na UV mapy.

Projekce je jednoduchá metody, která aplikuje texturu na objekt ve směru projekční plochy, ať už se jedná o pohled kamery nebo normálový vektor nějaké plochy či jiného objektu. Tento postup je vhodný pro přidávání detailů na již otexturované objekty a v případě jednoduchých polygonů i na kompletní texturování. Projektování také používají NURBS povrchy jako základní postup aplikace textur.

Pro polygony je více použitelná právě metoda aplikování textur na UV mapu, která reprezentuje rozmístění vrcholů polygonů ve dvourozměrném prostoru. Pokud si umělec uloží tento prostor jako podklad při vytváření textury, tak může jednoduše aplikovat barevnou informaci na dané místo mezi vybranými body polygonu, dodávající detaily podle toho, kde jsou právě potřeba.

Procedurální textury jsou takové, které vznikají matematickými výpočty a jsou náhodné podle parametrů. Lze s nimi dosáhnout skvělých detailů v podstatě v nekonečném rozlišení. V tom je největší potenciál těchto textur. Nevýhodou je pak vyšší výpočetní náročnost. Nelze je také efektivně použít na objekty, kde chceme specifické detaily v dané oblasti.

Klasické bitmapové textury jsou v tomto hledisku mnohem flexibilnější. Omezují nás sice malými detaily a artefakty při velkém přiblížení, ale dovolují přidávat detaily tam, kde jsou zrovna potřeba a jsou jednoduché na výpočet po aplikaci na objekt. Bude- li se umělec držet základních pravidel při práci s bitmapovými texturami, tak dávají vynikající výsledky. Jedno z těchto pravidel je tvorba textury s alespoň dvojnásobným rozlišením, než v jakém se bude vyskytovat na renderu.

2.1.3 Normal mapping

Většina modelů v hale budovy A má dodané detaily použitím normálových map. Jejich popis najdete v tématu „Normal mapping". Použití těchto map zvětšuje efektivitu práce. Nemusel jsem vytvářet takové množství polygonů k dosažení detailů, tedy se snížila náročnost aplikace na paměť a zvedla se rychlost prostředí. V ohledu na čas je vytvoření detailu na objektu pomocí normálové mapy nesrovnatelně jednodušší. Ale neplatí to všeobecně, protože některé detaily jsou modelované lepší než ty, které jsou vytvářené texturou.

2.1.4 Osvětlení

K nasvětlení jsem použil standardní světla, většinou bodová světla a plošná světla s jemnými stíny a vysokou hloubkou tracingu stínů. Tato světla se chovají podle fyzikálních rovnic. Mají kvadratický úbytek, přesně tak, jak je tomu i v reálné situaci. Velikým úskalím těchto světel je jejich výpočetní náročnost s rostoucím počtem jejich instancí. V situaci, kdy se ve scéně vyskytovalo třeba i čtyřicet světel, bylo skoro nemožné dostat render v rozumném čase. K tomu tyto světla jenom stěží simulují odražené světlo, tak bylo potřeba zapnout i final gather, tedy algoritmy simulující odražené světelné paprsky, s nimiž se renderovací čas značně zvyšuje.

Final Gathering může na druhou stranu renderování usnadnit. V mém případě jsem zkusil nahradit skoro dvě třetiny standardních světel geometrií, která není ve výsledku vidět, ale produkuje levné final gather světlo a dodává scéně na uvěřitelnosti.

Podporou pro realističtější osvětlení pak je GI (global illumination). Ten byl produkován photony vystřelujícími z plošných světel simulujících světlo přicházející z okenních ploch.

2.1.5 Rendering

V první kapitole jsou popsány moderní metody zpracování počítačové 3D grafiky a v mé vlastní realizaci jsou použity téměř všechny výše zmiňované metody. Problémová byla především hloubka ostrosti (depth of field), která nešla použít vzhledem k postupu jakým je renderováno finální panorama.

2.1.6 Tvorba panorama

Vytvořit panorama lze několika způsoby. Nejznámější je skládání fotografii (renderů) pomocí specializovaných 2D nástrojů. Tyto nástroje jsou ve většině zpoplatněné a pro mnohé nedostupné. S trochou zdatnosti je možné používat i běžné editory na grafiku, ale při použití velkého množství obrázků jsou výsledky nezaručeny.

Ve 3D prostředí je možné jednoduché východisko. Pokud software poskytuje možnost mapovaní textur na pozadí do bitmapy, pak je nejjednodušší pracovní postup následující:

Nejprve si umělec určí kde bude střed panorama. Do tohoto středu se vloží koule se standardním UV mapováním, které deformuje plochy jednotlivých polygonů směrem k hornímu a dolnímu konci mapy. Na tento materiál se následně mapuje materiál s následujícími vlastnostmi. Pohlcené světlo je nulové, odražené světlo je na maximu, tento objekt nesmí vrhat stíny a barva je černá, i když v tomto případě to je jedno, protože nebudou vidět nic víc než odrazy.

V nastavení rederovacího enginu má Maya možnost batch render do textury, kde se nastaví zapečení světla, barvy a odrazů (s větší hloubkou reytracingu je lepší) do textury zabírající plochu kompletní UV mapy koule. Výsledný obrázek je dokonalé 360-stupňové panorama určené na kouli. Samozřejmě je možné toto panorama vytvořit i na krychli nebo

polokouli (koule, polokoule a krychle jsou tři základní projekce na panorama). Dle mého názoru nejlepší výsledky dává panorama na kouli.

Úskalím takového renderování panorama na pozadí, je výpočetní náročnost a požadavky na paměť. S průměrným nastavením batch zapečení textury (uložení kompletní barevné informace do souboru) na kouli o rozlišení 4096x2048 pixelů bylo potřeba mít alespoň 8GB paměti v počítači a i tak bylo výsledné renderování hotovo za 36 hodin.

2.1.7 Panorama užití

Cílem kompletního procesu tvorby 3D scény zachycující halu budovy A bylo několik. Prvním z nich byla designová úloha. Úvodní obrazovka aplikace je interaktivní panorama umožňující zábavný a přitom přehledný systém práce s informačním systémem. Jako podklad by samozřejmě bylo možné použít fotografický materiál spojený ve sférický 360 stupňový obraz, ale vzhledem k povaze designu aplikace byla 3D reprezentace vhodnější.

Výstupem dalších částí aplikace jsou 3D průlety budovou v jednotlivých patrech. Jsou to sekvence obrázků o počtu nepřesahujícím 70 snímků kvůli náročnosti na internetové připojení. Průlety jsou v některých částech dlouhé a tak má menší počet snímků za následek vysokou rychlost průletu, ale orientace v budově je zachována.

3 Aplication workflow

3.1 Technologie

3.1.1 PHP

3.1.1.1 Php obecně

PHP je široce používaný skriptovací programovací jazyk původně navržený pro tvorbu dynamických webových stránek. Z tohoto důvodu je PHP kód začleňován nejčastěji v HTML, XHTML či WML formátu a je překládán na straně serveru s instalovaným PHP modulem, který následně produkuje výsledný HTML dokument. PHP program je prováděn interpretem často v příkazové řádce na straně serveru a uživateli je doručován výsledek operace ve standardním výstupu. Může také sloužit jako grafická aplikace. Php je dostupný na většině moderních webových serverech a funguje na většině operačních systémů a výpočetních stanicích.

Syntaxe jazyka je inspirována několika programovacími jazyky jako jsou Perl, C, Java a Pascal. Podporuje mnoho knihoven pro různé účely jako je zpracování textů, práce se soubory, grafika a práce s databázovými systémy. Nejběžnější je pak spojení PHP a MySQL, ale podporuje i systémy ODBC, Oracle a jiné. Podporu zajišťuje PHP pro velké množství internetových protokolů (HTTP, SMTP, SNMP, IMAP, POP3,...).

PHP v sobě sdružuje několik programovacích jazyků, je jednoduchý a dává vývojářům určitou svobodu ve tvorbě syntaxe. V kombinaci s výše zmíněným MySQL a servery Apatche, pak dává kompletní nástroj pro vývoj malých až velkých webových aplikací. Všechny tyto vlastnosti jsou důvodem pro velkou oblibu tohoto skriptovacího jazyka.

3.1.1.2 PHP v aplikaci

Flexibilita, kterou poskytuje tento programovací jazyk, je jeden z důvodů k výběru právě PHP pro realizaci interaktivní 3D navigace v budově A. Je to ale bezchybný workflow s databázovými systémy MySQL a schopnost jednoduchého generování XML výstupů, které jsou hlavním důvodem výběru PHP.

Hlavní využití PHP spočívá ve vytvoření jednoduchého a spolehlivého administračního systému pro editaci dat v databázi a zachování použitelnosti aplikace na delší časové období. Dalším důležitým úkolem PHP je získávání informací z databázové struktury a vracení XML souborů pro vypisování výsledků.

3.1.2 MySQL

MySQL je relační databázový systém běžící jako server a poskytující víceuživatelský přístup k danému konkrétnímu počtu databází. Tento systém je vyvinut pod záštitou veřejné licence a je dostupný zdarma všem uživatelům. Samotný systém MySQL byl vyvinut švédskou firmou MySQL AB a nyní je majetkem Sun Microsystems, pobočkou Oracle Corparation.

MySQL naleznete ve velkém množství projektů pracujících s databázemi. Popularita tohoto systému je také široce spojena s oblíbeností PHP, který je často používám právě ve spojení s MySQL. Jedni z velkých zástupců webových portálů založených na MySQL jsou Flick, Facebook, Wikipedia a YouTube.

MySQL je multiplatformní systém podporující širokou paletu systémových platforem včetně AIX, FreeBSD, Linux, Novell NetWare, Mac OS X, Windows a řadu dalších.

3.1.2.1 Management a grafická uživatelská rozhraní

MySQL je relační databáze založená na mnohdy složitých vztazích a proto je vyvinuto mnoho grafických uživatelských rozhraní (GUI) pro usnadnění práce s daty

v databázi. Uživatelé mohou používat nástroje v podobě příkazových řádků nebo plných softwarových rozhraní.

Oficiální rozhraní pro práci s MySQL databázemi je MySQL Workbench. Tento nahrazuje původní nedokonalé GUI nástroje a poskytuje široké možnosti práce s databází. Pomocí této aplikace lze vytvářet design a model databáze, vyvíjet sql dotazy a administrovat databázi. MySQL Workbench je dostupný ve dvou verzích a to standardní free verze, open source verze upravená komunitu a také komerční verze, která rozšiřuje a upravuje možnosti z komunitní verze.

Existuje množství „third patry" verzí GUI pro správu MySQL databází. Patří sem např.: phpMyAdmin, HeidiSQL, Navicat a jiné.

3.1.2.2 Nasazení MySQL

Přestože MySQL začalo jako jednoduchá alternativa k výkonným a široce používaným databázím, začalo se vyvíjet a dnes již splňuje požadavky i pro velké databázové systémy. Všeobecně je používáno v malých a středně velkých databázích na jednom serveru, buď jako součást webového portálu nebo jako samostatný databázový server. Popularita MySQL tkví v jeho jednoduchosti a snadném použití spolu s intuitivním editačním rozhraním, kde nejpopulárnější je phpMyAdmin. V případě použití ve středních aplikacích lze posílit MySQL multiprocesorovým hardwarem a větším množstvím paměti.

Přesto existují limity pro samostatné servery co se týče použití Mysql. Pro velké aplikace je zapotřebí více serverů. Často se používá systém, kdy máme jeden hlavní server pro operace zápisu, který se replikuje na množství podřízených serverů, které v případě nutnosti, zastoupí funkci nadřízeného serveru a tím zabraňují zpomalení a logům na straně serverového systému.

3.1.3 Flash

3.1.3.1 Flash Player

Adobe Flash Player je software pro přehrávání multimediálního obsahu vytvářeného pomocí aplikace Flash a jeho programovacího jazyka Actionscript. Jedná se o široce rozšířený plugin vyvinutý společností Macromedia a v současnosti vyvíjen a distribuován koncernem Adobe po její akvizici. Flash Player spouští soubory s příponou .swf vytvářené pomocí Flash authoring nástrojů jako jsou Flash CS3 authoring nebo Flex Builder 2.

Flash Player má v sobě zakomponovanou podporu skriptovacího jazyka Actionscript založeném na ECMA skriptu. Původně byl vytvořen pro prezentaci dvou dimenzionálních vektorových animací a od té doby se rozrostl až na nástroj pro „rich Internet aplications". Flash Player je dostupný jako plugin pro všechny moderní internetové prohlížeče.

3.1.3.2 Actionscript 3.0

Actionscript 3.0 je nový a rozdílný a zároveň překvapivě známý těm, co někdy pracovali s verzí Actionscript 2.0. AS 3.0 je nový především z toho důvodu, že je postavený od úplných základů znovu na bázi nové edice ECMA, a používá novou virtual machine (AVM2). Přes celkový nový dojem, je i nadále dostatečně podobný Actionscriptu 2.0, tudíž lidé, kteří pracovali se starší verzí tohoto programovacího jazyka, nemají příliš velké problémy s přechodem na AS 3.0.

3.1.3.3 Co je Actionscript 3.0

Většina lidí má jistou představu o tom co znamená Actionscript, jistě však další přiblížení udělá programovací jazyk ještě jasnějším. Actionscript je programovací jazyk k tvorbě obsahu pro Flash Player (viz. výše). Pro tvorbu tohoto obsahu je možné používat nástroje jako Flash CS3, Flash CS4 a nově vyvíjený CS5 balík, dále pak Flex Builder za použití nástrojů jako jsou kreslící nástroje, knihovny symbolů, časové osy a MXLM. Actionscript lze použít jako doplněk těchto nástrojů nebo jako jejich náhrada, kdy je tento obsah vytvářen programově. Tato možnost je mnohdy lepším řešením. Actionscript je často nezbytný pokud je potřeba vytvořit vysoce dynamické, interaktivní,

znovupoužitelné nebo uživatelsky upravitelné aplikace. Actionscript a s ním spojené nástroje lze použít například pro:

- načítání obrázků
- přehrávání zvuku a videa
- programově vytvářená grafika
- načítání dat jako jsou XML soubory
- reagování na uživatelské události jako jsou „kliknutí myši" nebo „najetí nad daný objekt"

3.1.3.4 Kde lze použít Actionscript?

V současnou chvíli jsou možné dva hlavní způsoby jak pracovat s Actionscriptem. A to vytváření flashových aplikací pomocí Flash authoring (Flash CS3, CS4) nebo se používá Flex a Flex Builder 2. Samozřejmě není dáno, kdy a kde používat authoring nástroje nebo Flex. Spíš se použití těchto způsobů prolíná. Pokud se bude umělec rozhodovat, co použít, je dobré vědět, jaký pracovní postup bude použit. Ti co raději používají kreslící nástroje a časové osy, sáhnou nejspíše po Flash authoring nástrojích (Flash CS3 a vyšší). Naopak, Flex Builder 2 je dobrý pro ty, kteří používají raději samotný programovací jazyk v robustním IDE (integrated development environment – integrované vývojové prostředí) a přijde jim nepohodlné používat animační metafory nabízené Flash CS3 a jinými. Dobrá zpráva je, že AS 3.0 se neliší v závislosti na použitém vývojovém prostředí.

Pokud vývojář aplikace používá Flex Builder 2, tak je povinen používat striktně pouze objektově - orientovaný design znamenající, že hlavní vstupní bod do aplikace musí být class (třída). Tak to však neplatí pro Flash authoring nástroje, kde je možnost pokládat actionscript kód na jednotlivé klíčové snímky na časové ose a programovat se zde dá procedurálně i objektově skrze třídy. Samozřejmě to nese i své nevýhody a to možnost pozdějšího upravování kódu. Pokud jsou části kódu rozesety po různých snímcích, je pak složitější se dohledat chyb, které se mohou vyskytnout a pro někoho, kdo bude používat skripty po nás, je těžké se dobře a rychle zorientovat.

K efektivnímu zjednodušení programovacího kódu je dobré mít rozsáhlé okomentování řádků programu a přehledné pojmenování proměnných a podobně.

3.1.3.5 Novinky v Actionscriptu 3.0

Jak jsem již zmínil dříve, AS 3.0 obsahuje mnoho nových vlastností. Následuje stručný výběr těch nejzásadnějších.

Display List

V Actionscriptu 2.0 se vyskytovaly tři základní typy objektů, které mohly být zobrazeny: movie clips, buttons a text fields (video klipy, tlačítka a textová pole). Tyto typy nedědí od stejného zdroje a to znamená, že polymorfismus u těchto typů nefunguje. Kromě toho, instance těchto zobrazovacích typů měly vždy přesně daný parent-child (rodič-potomek) vztah s ostatními instancemi. Například při vytváření movie clipu se muselo vytvářet movie clip jako potomek dalšího existujícího movie clipu. Nebylo tak možné přesouvat tento klip z jednoho rodiče na jiného.

V Actionscriptu 3.0 je mnoho nových display typů. Jako přírůstek ke zmiňovaným třem typům jsou zde i shapes (tvary), sprites (pohyblivé symboly), loaders (načítač), bitmaps (bitmapy) a mnohé další. Všechny display typy vycházejí z jednotného zdroje flash.display.DisplayObject. Poskytuje to možnost použití polymorfismu. Kromě toho je asi nejvýznamnějším pokrokem, že mohou být display objekty použity bez závislosti na jiných display objektech, a mohou být potomkem různých objektů a přesouvány z jednoho rodiče na jiného. Jinými slovy lze vytvořit např. textové pole v AS 3.0 jednoduše voláním konstruktoru a toto textové pole bude nezávislé na jakémkoli jiném objektu.

```
var text:TextField = new TextField();
```

Poté je už jenom formalita připojit tento objekt k nějakému kontejneru. Jako příklad připojím toto textové pole kontejneru, který může být jakýkoli jiný display objekt:

```
kontejner.addChild(text);
```

Runtime errors

Actionscript 3.0 poskytuje mnoho nových běhových chyb. Je to velice důležitá novinka, která umožňuje rychlejší diagnostiku problémů ve skriptu. V AS 2.0 některé chyby probíhaly v naprosté tichosti a bylo tak mnohem složitější případný problém odhalit.

Runtime datové chyby

Striktní otypování bylo v AS 2.0 použito pouze kompilátorem a ne za běhu. V AS 3.0 jsou striktní datové typy přítomny i za běhu a tak se neshodování přiřazovaných typů za běhu objeví jako chyba. Výsledkem odstranění těchto chyb je lepší práce s pamětí a zachování konzistence datových typů.

Method Closures

V Actionsriptu 3.0 mají všechny metody uzavírací systém, znamenající, že každá metoda vždy obsahuje referenci na objekt, ze kterého je původně volána. Toto je důležité pro ovládání událostí a stojí v ostrém kontrastu s uzavíráním metod v AS 2.0. Při vytváření referencí na metodu v Actionscriptu 2.0, objekt, ze kterého je metoda referencovaná se nezachovával. Toto způsobovalo mnoho problémů při vytváření event listenerů. V Actinscript 3.0 je toto vyřešenol.

Pravý model událostí

V Actionscriptu 3.0 je model událostí zabudován přímo do samotného jádra programovacího jazyka. Flash.events.EventDispatcher je základní třída pro mnohé základní třídy jazyka AS, display objekty nevyjímaje. Znamená to, že je zde pouze jedna možnost, jak vyvolávat události.

Regulární výrazy

Regulární výrazy jsou mocný nástroj jak vyhledávat výsledky podle vzorů. I když jsou regulární výrazy již dlouhodobou součástí sesterských jazyků jako je např. JavaScript, je přidám do Actionscriptu právě s verzí 3.0. AS obsahuje třídu RegExp, která slouží pro práci s regulárními výrazy ve Flash Playeru.

E4X

E4X je zkratka pro ECMAScript pro XML a je novým způsobem jak pracovat s XML daty v Actionscriptu 3.0. Samozřejmě je možné pracovat s XML také původním způsobem z předchozích verzí AS.

3.1.3.6 Actionscript v aplikaci

Interaktivní navigace v budově A je z velké části postavená právě na Actionscriptu a to verzi 3.0, která, jak je zmíněno výše, nabízí rozsáhlejší možnosti než předchozí verze, Osobní názor na AS 3.0 je takový, že i pro člověka, který pracoval ve větším měřítku s verzí 2.0, je přechod na novou verzi značně komplikovaný, protože se v postatě jedná o nový programovací jazyk. Nějaké podobnosti samozřejmě existují, ale hlavní funkčnost AS 3.0 chce nový postup myšlení. Po zaučení s Actionscriptem 3.0 jsem musel uznat, že je práce mnohem všestrannější a efektivnější. Například ovládání událostí, jednoduchost vytváření display objektů a další, usnadňují vývojáři život.

Aplikace je postavena na nástroji Autodesk Flash CS3 s použitím kreslících nástrojů, knihovních prvků a většinou procedurálního programování s malým použitím tříd. Některé komponenty jsou grafické s vlastní identifikací a jsou vložené jako součást načítaného obsahu v kořenové části aplikace a jiné jsou generované dynamicky programováním podle potřeby. Například veškerý obsah sekce "novinky" je programově generovaný. Programové generování snižuje datovou velikost kořenového souboru aplikace. Samozřejmě dynamičnost aplikace se tak zvětšuje.

K vytvoření aplikace je využito všech novinek, které přinese Actionscript 3.0. Velký důraz je dán na display objekty a ovládání událostí. Také je v aplikaci hojně použit postup práce s externími daty, ať už jsou ze souborů XML nebo generované pomocí PHP

z databázového systému. Tento workflow bude více ukázán dále v textu i s ukázkami kódů a představením jejich funkčnosti.

Aplikaci potom nebude uživatel spouštět v nějakém swf přehrávači, ale ve standardním webovém prohlížeči. Podkladem pro aplikaci bude tedy HTML, kam se vloží výsledná flashová část. Z tohoto základu bude probíhat komunikace s php, mysql a actionscriptem.

3.2 Datový koloběh

3.2.1 Životnost aplikace

3.2.1.1 XML souborový systém

Při vývoji této aplikace bylo nutné mít na paměti její znovupoužitelnost. Mnoho podobných webových portálů psaných v Actionscriptu je založeno na čistém XML souborovém systému bez vlastního editačního zázemí pro XML. V podobném duchu byla vytvářena i aplikace pro Informační Centrum TUL a aktualizace se tak setkává s problémy, protože lidé, kteří se o aktualizaci mají starat, neznají vnitřní zákonitosti xml souborů. Pro aktualizaci takového jednoduchého xml je potřeba vědět, co znamenají jednotlivé tagy, ale i když jsou tyto tagy pojmenovány příhodně, může to i tak být problém pro neinformatické správce dat.

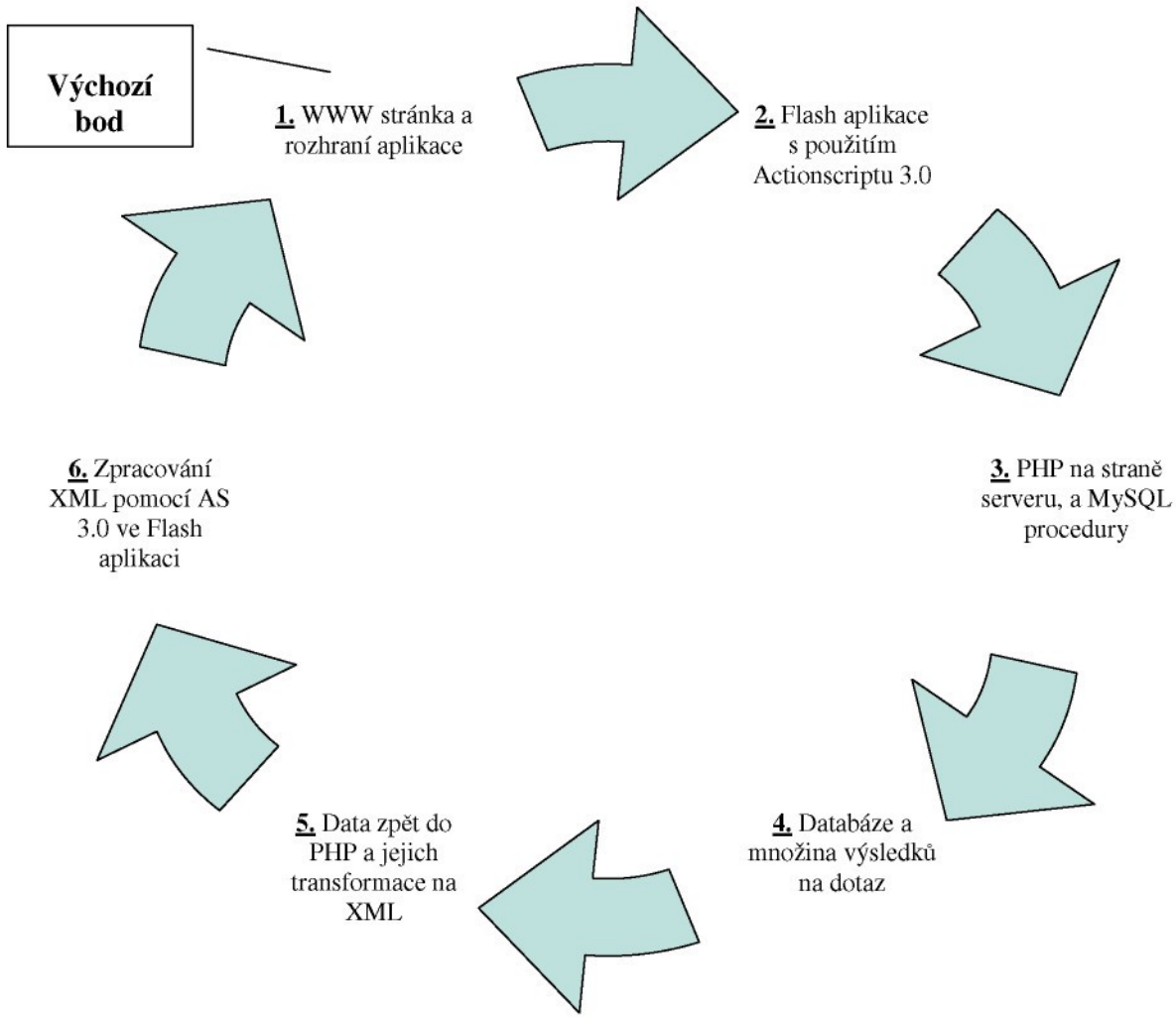
3.2.1.2 XML generované pomocí PHP a MySQL

Pro tuto aplikaci jsem zvolil systém s delší životností a menšími nároky na znalosti lidí, kteří se budou starat o aktualizaci aplikace. Systém je založen na kombinaci XML, kterému dobře rozumí Actionscript a PHP s kvalitní podporou příkazů pro práci s databázemi MySQL. Ve výsledku vzniká proces, o který se může starat člověk s minimální znalostí práce s počítačem. Podrobněji se o jednotlivých částech systému pro lepší životnost a dlouhodobost zmíním v dalších kapitolách.

3.2.1.3 Snadná grafická obnovitelnost

Interaktivní navigace v budově A je z velké části založena na průletových animacích. Tyto animace jsou sekvence obrázků seřazených za sebou a přehrávaných podle potřeby uživatele. Každá animace průletu je vytvořená pro jednotlivé důležité body v budově, ať už se jedná o učebnu nebo místnost se zaměstnanci. Tento obsah je načítán na základě dat uložených v databázi a to konkrétně podle čísla místnosti. Každá místnost v budově má své jedinečné označení, které udává na jakém patře se nachází a o jakou místnost v pořadí se jedná. Ke každému tomuto záznamu je přiřazena animace složená ze sekvencí obrázků s relevantním obsahem. V případě potřeby je velice jednoduché změnit právě danou animaci v jakémkoli editačním programu a tak se dosáhne aktualizace požadované části, bez nutnosti zasahovat do struktury aplikace, která není jednoduchá. Toto je další ze způsobů, jak udržet aplikaci co nejaktuálnější s minimálním úsilím a znalostmi.

3.2.2 Kruh dat



3.2.2.1 1. část aplikační rozhraní ve webovém prohlížeči

Pro uživatele je samozřejmě nejdůležitější webové rozhraní aplikace. Poskytuje přístup k dalším funkcím aplikace. Uživatel přistupuje k portálu a podle jeho činnosti se aktivují další články řetězce funkcí, které následně dodají relevantní data. Běžný způsob práce s předkládanou webovou aplikací je vyhledávání, procházení budovou A a zobrazování novinek a jiných zajímavých "oblastí". Z aplikačního rozhraní jsou posílány proměnné, které jsou následně zpracované flash aplikací na bázi Actionscriptu 3.0 a posílány dále php procesům na straně serveru. Odpověď je vyhledána v databázi a poslána

zpět jako výsledek PHP, který generuje XML soubor posílaný zpět do Flashe a aktivující procedury pro jeho zpracování. Výsledek je zobrazen jako textová data v aplikaci. Podrobněji rozeberu jednotlivé fáze v následujícím textu.

3.2.2.2 2. Flash a Actionscript 3.0

Celá navigace v budově A je založena na technologii Adobe Flash. Obsah je tak mnohem interaktivnější než je tomu u běžných webových stránek, i když moderní pluginy jako je Java, dávají nové zajímavé možnosti jak osvěžit statický vzhled webových stránek. Úkony, které uživatel vykoná, jsou založeny na funkcích a procedurách programu Actionscript. Zadá- li uživatel text do vyhledávacích oken a zvolí odeslání, aplikace najde příslušnou funkci a zpracuje data do formy, kterou je možné bezpečně odeslat ke zpracování do PHP skriptu. Většina dat je odesílána metodou „Post".

3.2.2.3 3. PHP na straně serveru

Po tom, co se odešlou data z Flashe, jsou načtena metodou „post" a přiřazena do proměnných. Následně proběhne PHP skript, který ve většině případů zavolá MySQL procedury a čeká na data z databáze. V této fázi se vytváří MySQL dotazy se všemi omezeními. Tyto omezení jsou různé podle požadovaných dat, které uživatel potřebuje.

3.2.2.4 4. Databáze a množina dat

Na základě požadavků MySQL se v datové struktuře databáze naleznou a zařadí relevantní data. Datová struktura v této aplikaci není příliš složitá a i z těchto důvodů je použití MySQL dostačující.

3.2.2.5 5. Zpracování dat a PHP generování XML

Pro flashovou aplikaci je nejběžnější typ datových souborů právě XML a s příchodem Actionscriptu 3.0 a E4X procedurami je jeho zpracování ještě jednodušší. Samozřejmě pokud se pošlou data z PHP jako parametry anebo jako strohý string, tak to také umí flash zpracovat. Z vlastní zkušenosti mi přijde nejefektivnější způsob práce právě s XML. Z těchto důvodů jsem se rozhodl generovat výstup z PHP jako XML. Generování

probíhá stejně jako při komunikaci běžného HTML webu a PHP s těmi rozdíly, že při vypisování výsledku PHP skriptem se vypisují také XML tagy. Pokud otestujeme jenom dané PHP v prohlížeči, můžeme si nechat vypsat strohý text obsažený v XML. Přehlednost je mnohem lepší než u HTML protože tagy, které zvolíme při vypisování XML, si můžeme nazvat podle potřeby. Také máme volnost ve způsobu, jestli data vkládáme do XML jako atributy nebo jako string obsah v tagu. Příklady generovaných výsledků zmíním v dalších kapitolách i se vzory php, které to mají na starosti.

3.2.2.6 6. XML do Actionscriptu

Výsledné XML zpracovávám pomocí E4X procedur, které dovolují ohromně jednoduchou práci s XML soubory. Podle toho, jaká data si uživatel vyžádal, vypadá i soubor, který je předán flashi k zobrazení. Je-li potřeba vypsat nějakého zaměstnance univerzity, budou tagy v XML souboru tomu přizpůsobeny. Můžeme pak vypisovat do textových polí například takto:

```
DisplayObject.zamestananec_jmeno_txt.text = mojeXML.zamestnanec.jmeno;
```

Je to jednoduchý a velice efektivní způsob jak data dostat na místa, kde je potřeba. Uživatel tento koloběh samozřejmě nevidí. Zaznamená to pouze změnu v rozhraní aplikace a pokud vše funguje jak má, nedochází k chybám na straně aplikace a serveru.

3.3 Databázový model

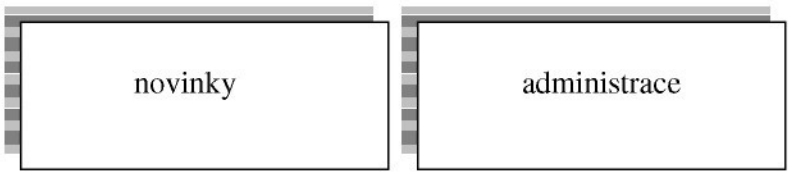
3.3.1 Typ databáze v závislosti na aplikaci

Databáze, kterou jsem navrhl pro Interaktivní navigaci v budově A se odvíjí od zadaných požadavků, které má systém zpracovávat. Aplikace se řadí mezi malé a středně velké DB aplikace a podle toho je i dimenzována databáze. Tabulky v databázi jsou vytvořené pro proměnná data a také pro zaznamenání dat stálých s možným růstem obsahu.

Mezi jednotlivými tabulkami jsou definované relační vazby, které jsou ošetřeny pomocí PHP při vytváření záznamů vkládaných do databáze. Tyto vazby nejsou nikterak složité a zajišťují správnou aktualizaci dat v tabulkách, které jsou vzájemně propojené.

Při editaci zaměstnanců je administrátor požádán o doplnění místnosti, ve které tento zaměstnanec sídlí. Po vybrání odpovídají místnosti ze seznamu, se automaticky připojí data o patru, typu místnosti, jejím názvu a lidech, kteří jsou již v této místnosti zařazeni. Je to jednoduchý, ale přesto funkční systém aktualizace dat v databázi pomocí PHP.

3.3.2 Struktura databáze

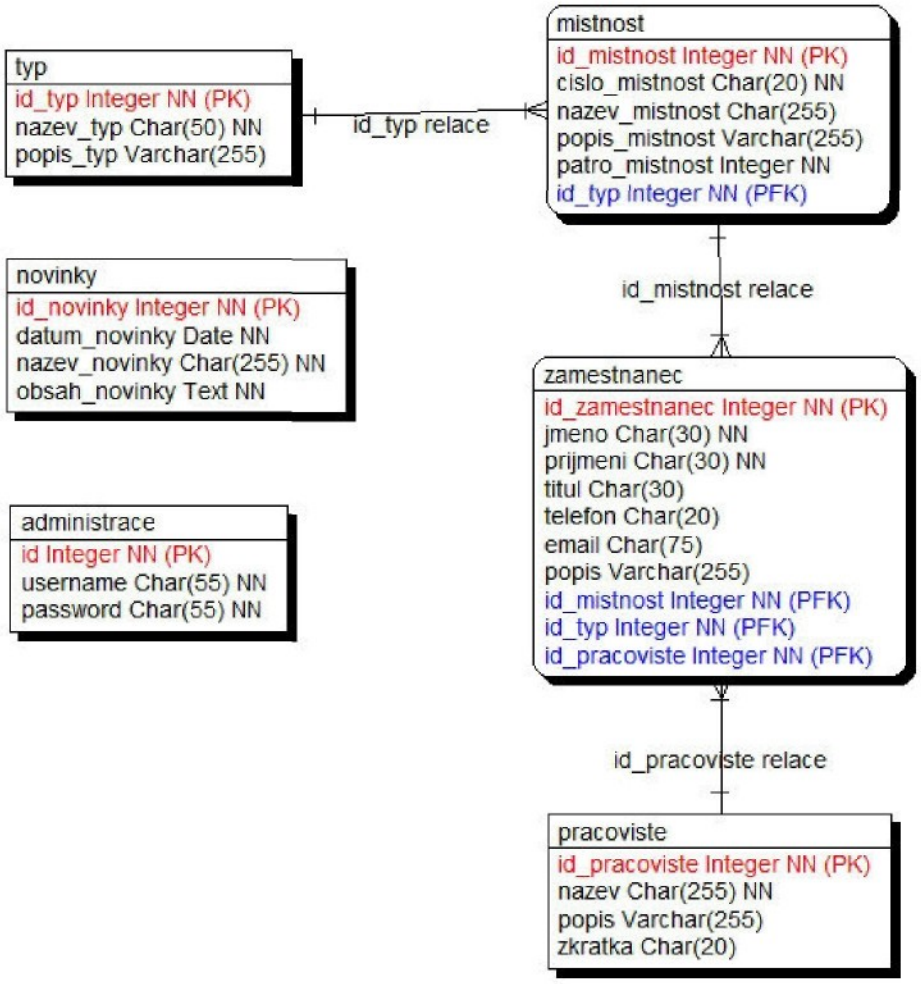


Nezávislé tabulky souží pro uchování aktuálních dat a pro možnost vyhledávání starších záznamů.

Tabulka „pracoviště“ obsahuje popis jednotlivých pracovišť, kanceláří, laboratoří a jiných zařízení, které jsou v budově A zastoupeny. Jejich definice je předávána pomocí identifikačních klíčů každému zaměstnanci. Umožňuje se tak návštěvníkovi aplikace možnost vyhledávání pomocí rozličných parametrů.

Tabulka typ místnosti dále specifikuje, jestli se jedná o učebnu, počítačovou místnost, laboratoř, kabinet a jiné. Použitelné jsou tyto údaje při vyhledávání a poskytují rozšířené informace o daném místě či zaměstnanci.

Obě tyto podpůrné tabulky jsou podkladem pro rozšířené informace o zaměstnancích



obr. 1 struktura databáze

a místnostech. Vzhledem k použití této aplikace a jejímu rozsahu by bylo zbytečné vytvářet složitější strukturu databáze. Samozřejmě je zde potenciál na případné rozšiřování aplikace o mnohá další data a o pokročilejší systémy, ale to je úkol, jehož realizace je nad síly jednoho člověka. Aplikace by pak vstoupila do oblasti těch větších databázových systémů a bylo by potřeba početnějšího týmu lidí na realizaci jednotlivých cílů.

3.3.3 Databázový diagram

Databázový diagram říká jaké jsou vztahy mezi jednotlivými prvky diagramu. Mezi tabulkami jsou naznačeny vztahy pomocí relací, které jsou jedna ku jedné nebo 1:N. (tedy, že prvek z jedné tabulky může mít několik prvků druhé tabulky). Dá se z něho vyčíst, jakým způsobem budou přiřazovány prvky z tabulek. Například každý zaměstnanec může mít pouze jedno pracoviště a na jednom pracovišti může může pracovat "mnoho" zaměstnanců. Podobným způsobem lze přistupovat i k dalším tabulkám.

3.3.4 Databáze a animace

Jednou z důležitých vlastností této aplikace je možnost nechat uživatele procházet budovou pomocí renderovaných sekvencí. Tyto sekvence jsou v serverové struktuře ve svých vlastních adresářích a jsou pojmenovány tak, aby se vázaly ke správnému číslování místností v budově A. Toto číslování je zaneseno již v plánech budovy, tudíž se nemůže (nebo spíše by se nemělo) stát, že bude animace přiřazena špatné místnosti. V databázi jsou uložena data v tabulce místnosti. Každá místnost má přiděleno jednoznačné identifikační číslo, které se shoduje s názvem animované sekvence. Tuto sekvenci uživatel vyvolá pomocí rozhraní aplikace. Po komunikaci se serverem se dynamicky načtou potřebná data, xml a animace.

3.4 Papervision3D knihovny

Papervision3d je knihovna open-source tříd a procedur, které tvoří jednotný framework (kompletní sytém všech procedur a kódů přístupných z jakéhokoli místa v kódu s možností mnohonásobného použití jednotlivých částí) pro zpracování 3D simulací v prostředí Flash. Tyto knihovny jsou navrženy tak, že výborně spolupracují s Actionscriptem 3.0. Odpovídá tomu i způsob, jakým jsou knihovny programovány.

Všechny potřebné AS 3.0 skripty jsou zpracovány v Actionscriptu 3.0 v systému tříd, využívající všechny nové dovednosti tohoto programovacího jazyka.

Papervidion3d balík obsahuje i překvapivě komplexní skripty pro práci s reálnými 3D modely, které lze vytvořit v některém z dostupných 3D programů. Vývojář tak dostává skrze implementaci těchto knihoven možnost vytvářet ještě lepší aplikace a webové stránky, které můžou do jisté míry nahradit jiné pokročilé metody 3D. Možnosti při využití těchto knihoven jsou široké, počínaje jednoduchými kubickými objekty pro interaktivní menu nějaké webové stránky a kompletní trojrozměrnou hrou konče.

3.4.1 Základní knihovní prvky

Pro tvorbu zajímavých a uvěřitelných 3D efektů ve Flashi je potřeba různých skriptů, které zajišťují požadované efekty. Jsou to kamery, scény, rozhraní, základní geometrické tvary, materiály, osvětlení, renderovací enginy a podobně.

3.4.1.1 3D kamera, 3D scéna a 3D rozhraní (viewport)

Chceme-li simulovat trojrozměrný prostor pomocí Actionscriptu 3.0 a papervision3d, je potřeba založit standardní 3D prostor, který známe z grafických rozhraní jako je Maya a 3DsMax. Základem je viewport, který nám říká, kde se bude odehrávat celá simulace. Do rozhraní je potřeba umístit klasickou scénu, která bude určovat místo, kde se nacházejí požadované objekty, se kterými pracujeme. Představme si to jako kontejner, do kterého vkládáme objekty. Těchto kontejnerů můžeme mít libovolné množství (samozřejmě s ohledem na hardwarové vybavení). Samotný viewport a scéna nám ale neposkytují místo nebo objekt, kterým všechno snímáme. Proto je dalším krokem vytvoření kamery. Tato kamera má parametry a vlastnosti známé každému, kdo někdy měl možnost pracovat ve 3D aplikaci. Můžeme jí tedy rotovat, pohybovat, přibližovat jí nebo oddalovat, točit jí kolem objektu nebo svého těžiště a podobně. To všechno pomocí zapisování AS 3.0 kódu. Propojenost s knihovnami programovacího jazyka Actionscript nám umožňuje pomocí

vhodných událostí a standardních objektů, provádět tyto akce použitím myši, klávesnice a všeho ostatního co nám přijde zajímavé.

3.4.1.2 Objekty a zejména primitiva

Primitiva jsou základní geometrické útvary, které jsou implementovány uvnitř systému tříd balíku Papervision3D. Pokud vytváříme webové rozhraní a web aplikaci, pak jsou tím hlavním, k čemu programátor sáhne. V mém projektu jsem používal především koule a plochy pro reprezentaci požadovaných prvků (panorama a plány pater). Nebudu zacházet do přílišných detailů jakým způsobem jsou tyto objekty tvořeny, protože to je již popsánu v dokumentaci od vydavatele.

3.4.1.3 Materiály shadery a osvětlení

Chceme-li, aby trojrozměrné objekty reprezentovali nějaký obsah, pak jsou nejdůležitější materiály. V papervison3D knihovnách jich máme hned několik. Od základního drátěného (Wireframe) materiálu, přes bitmapové materiály, až po materiály tvořené flash swf videi, FLV videi až po streamované záznamy. Jejich variabilita dává veliký prostor pro fantazii a rozvoj vlastní kreativnosti. Materiály jsou kombinovatelné s shadery, které simulují různé druhy materiálů od matných po kovové a jiné další. Chceme-li vytvořit iluzi ještě větší dokonalosti, pak přicházejí na řadu světla. Zajímavé je, že i kombinací všech těchto prvků, je výpočetní náročnost stále na přijatelné hodnotě.

3.4.1.4 Renderer

Renderery jsou procedury, které zajišťují, aby se námi vytvořená 3D scéna spočítala a aktualizovala. Lze je volat pouze jednou nebo i mnohokrát za vteřinu, pokud je potřeba neustálé aktualizace polohy jednotlivých prvků figurujících ve 3D scéně. Příkladem toho je rotace panorama nebo pohyb plánu podle polohy myši.

3.4.1.5 Praktické zpracování 360 stupňového panorama

Následující příklad je funkční AS 3.0 kód, který se stará pouze o nastavení panorama a jeho rotaci. Jedná se o procedurálně zpracovaný programový kód, kde se

nevyskytují zápisy obsažené ve třídách. Takto je skript použitelný pouze na časové ose. Pokud bych chtěl, aby byl kód ještě univerzálnější, je nutné provést řádné úpravy do struktury třídy.

Nejprve si vložíme všechny potřebné komponenty z knihovny Papervision3D. Import knihoven používaných Acionscriptem není v tomto případě nutný, ale u vytváření definice třídy to je žádané.

```
import org.papervision3d.view.Viewport3D;
import org.papervision3d.cameras.Camera3D;
import org.papervision3d.scenes.Scene3D;
import org.papervision3d.materials.MovieAssetMaterial;
import org.papervision3d.materials.MovieMaterial;
import org.papervision3d.objects.primitives.Sphere;
import org.papervision3d.objects.primitives.Plane;
import org.papervision3d.render.BasicRenderEngine;
import caurina.transitions.*;
```

následuje úvodní definování základních proměnných , které jsou popsané výše.

```
var myViewport3D:Viewport3D;
var myCamera3D:Camera3D;
var myScene3D:  Scene3D;
var myMaterial: MovieAssetMaterial;
var mySphere: Sphere;
var myRenderer: BasicRenderEngine;
```

Nyní potřebujeme vytvořit nové instance předdefinovaných tříd. Použijeme tedy konstruktor „new“ a pokud je to možné, tak rovnou definujeme potřebné parametry. Papervision3D není na rozdíl od knihovny Actionscriptu 3.0 zabudován v programovacím prostředí a proto se neukazují potřebné hinty s nápovědy. Je tedy nutné často procházet zdrojovými kódy tříd knihovny Papervision3D nebo si otevřít nějakou dostupnou dokumentaci .

```
myViewport3D = new Viewport3D(1200, 700, true, true);
this.holder_panorama_mc.addChild(myViewport3D);

myScene3D = new Scene3D;
myCamera3D = new Camera3D;
myRenderer = new BasicRenderEngine;
myMaterial = new MovieAssetMaterial("panorama_mc",false);
```

```
mySphere = new Sphere(myMaterial,600, 50, 50);

myMaterial.doubleSided = true;
myMaterial.animated = true;
myMaterial.interactive = true;

myCamera3D.x = myCamera3D.y = myCamera3D.z = 0;
myCamera3D.focus = 400;
myCamera3D.zoom = 1;
```

Addchild metoda je nová v AS 3.0. Její popis se nachází v předchozích kapitolách. Následujícím zápisem přidáme objekt „mySphere“ do naší 3d scény.

```
myScene3D.addChild(mySphere);
```

Zapíšeme příkaz pro vyrenderování koule, tedy pro grafickou reprezentaci polohy koule, nastavení kamery a podobně.

```
myRenderer.renderScene(myScene3D, myCamera3D, myViewport3D);
```

Následující tři řádky ošetřují ovládání událostí vztahující se k 3D scéně. Události reagují na stisknutí myši, uvolnění myši a opuštění plochy aplikace.

```
stage.addEventListener(MouseEvent.MOUSE_DOWN, rotateCamera);
stage.addEventListener(MouseEvent.MOUSE_UP, rotateCameraStop);
stage.addEventListener(MouseEvent.MOUSE_OUT, rotateCameraStop);
```

Zde definujeme dynamicky vytvořenou grafiku, která se zobrazí při rotaci panorama na základě "posluchačů" událostí (eventListener).

```
var cl_stred:Shape = new Shape();
var cl:Shape = new Shape();
this.ukazatel_smeru_mc.enabled = false;
```

Funkce, které jsou níže popsané, pracují s dynamickým obsahem, který je součástí knihovny grafických prvků a 3D scénou, jíž jsme definovali na začátku.

RotateCamera funkce vytvoří tvar, konkrétně kruh vyplněný barvou, který se umístí doprostřed plochy a simuluje její střed. Tato funkce se spustí pouze v případě, že je stisknuto tlačítko myši. Nastaví se následně událost, opakující se každý snímek, tedy 25x za vteřinu.

```
function rotateCamera(e:MouseEvent):void {
      if (rotace_panorama == true){
            cl_stred.graphics.beginFill(0xffffff,alpha=1);
            cl_stred.graphics.drawCircle(600,350,10);
            this.pstred.x = 600;
            this.pstred.y = 350;
            addEventListener(Event.ENTER_FRAME, rotateMyCamera);
      }
}
```

Funkce opakující se pořád dokola, dokud je stisknuto tlačítko myši.

```
function rotateMyCamera(e:Event):void {
      this.p_dummyes_mc.x = -100;
      this.p_dummyes_mc.visible = false;
```

Zde zjišťujeme, kde se nachází ukazatel myši a výslednou hodnotu (ta je buď kladná nebo záporná, protože ji zmenšujeme o střed plochy, kde se s myší pracuje) dělíme hodnotou 150, abychom dostali malou hodnotu, kterou přičítáme k rotaci kamery.

```
      myCamera3D.rotationY += (mouseX - stage.stageWidth / 2)/ 150;
      myCamera3D.rotationX += (mouseY - stage.stageHeight / 2)/ 150;
      myRenderer.renderScene(myScene3D, myCamera3D, myViewport3D);
```

Podle polohy myši při aktivní události rotujeme i objekt šipky. Převádíme tak polohu na úhel v radiánech a pomocí goniometrických funkcí na úhel ve stupních. Tento výsledek vkládáme do rotace objektu.

```
      var theX:int = mouseX - 600;
      var theY:int = (mouseY - 350) * -1;
      var angle = Math.atan(theY/theX)/(Math.PI/180);
      if (theX<0) {
            angle += 180;
      }
```

```
        if (theX>=0 && theY<0) {
              angle += 360;
        }

        this.ukazatel_smeru_mc.rotation=(angle*-1) + 90;
        this.ukazatel_smeru_mc.x=mouseX;
        this.ukazatel_smeru_mc.y=mouseY;
```

Po každém provedení funkce potřebujeme vymazat grafická data, aby na ploše byl pouze jeden aktuální objekt.

```
        cl.graphics.clear();
        cl.graphics.lineStyle(2,0xffffff,0.5,false, CapsStyle.ROUND);
        cl.graphics.moveTo(600,350);
        cl.graphics.lineTo(mouseX, mouseY);
        addChild(cl);
}
```

Pokud uživatel pustí tlačítko myši, dojde ke zrušení události „rotateMyCamera" a k resetování polohy objektů z knihovny prvků a vymazání dynamických grafických objektů.

```
function rotateCameraStop(e:MouseEvent):void {
        if (rotace_panorama == true) {
              this.p_dummyes_mc.x = 600;
              this.p_dummyes_mc.visible = true;
              this.ukazatel_smeru_mc.x = 0;
              this.ukazatel_smeru_mc.y = -25;
              this.pstred.x = 25;
              this.pstred.y = -25;
              cl.graphics.clear();
              removeEventListener(Event.ENTER_FRAME, rotateMyCamera);
}
```

4 Aplikace

4.1 Administrace

Administrační rozhraní je vytvořeno se stejným záměrem jako u všech podobných aplikací vznikajících na internetu. Jde o to, aby uživatel a správce dat uložených v databázi, mohl jednoduchým způsobem aktualizovat data. Rozhraní je postaveno na internetových standardech HTML, PHP, MySQL.

V rámci administrace je možno upravovat novinky, vkládat nové a mazat nepotřebné, upravovat zaměstnance, jejich přiřazení do místností, emaily popisy a další. Celkový systém aktualizace dat je propojen se strukturou databáze, a nabízí možnost upravovat v podstatě všechna data, která databáze obsahuje.

Příkladem je vložení nového zaměstnance s podrobnostmi o něm. K tomu je nutné získat data z několika tabulek. Místností, typů místností a pracovišť. Osoba pověřená vložením dat nejprve vyplní formulář s osobními údaji zaměstnance, poté přiřadí místnost, kde bude usazen. K této místnosti je již automaticky přiřazen její typ (stalo se tak při vkládání místnosti do databáze). Dále je potřeba přiřadit pracoviště zaměstnance pokud je známo, pokud ne, automaticky je přiřazena Technická Univerzita v Liberci.

Ve výsledku se jedná o jednoduchou aplikaci, která má doplňující funkci pro interaktivní systém navigace v budově A. Programový obsah je v podobě PHP a MySQL. Podrobnosti zde nebudu uvádět a spíše se budu soustředit na programový obsah Actionscript 3.0 ve vlastní aplikaci.

4.2 Interaktivní 3D navigace v budově A

4.2.1 Seznámení s aplikací

Interaktivní 3D navigace v budově A je vytvořena jako průvodce se zabudovaným databázovým systémem. Uživatel je skrze aplikaci schopen najít různá data spojená s budovou A, ať už se jedná o místnost, různá pracoviště nebo jednotlivé zaměstnance. Požadovaná data lze potom ukázat vzhledem k místnostem na každém z pater této budovy. V praxi to znamená, že pokud uživatel hledá například email nějakého konkrétního zaměstnance, pak lze tyto data (pokud jsou zanesena v databázi) zobrazit na různých místech aplikace jako je rychlé vyhledáváni, podrobné vyhledávání, seznam místností na patrech nebo v animaci cesty k cílové lokaci. Se zobrazením hledaného emailu lze spustit i videa s lokací cílové osoby.

Samozřejmostí této aplikace jsou také aktualizované novinky týkající se Fakulty mechatroniky, informatiky a mezioborových studií, jejich zaměstnanců a jiné.

4.2.1.1 Aktuality

Aktuality jsou místo, kde lze zadávat nové události, změny v rozložení místností nebo i přemístění zaměstnanců. S rozsahem využití aplikace se rozšiřuje i záběr novinek a jejich obsah. Poslední tři novinky jsou vypisovány jako aktuality v boxech vpravo a pro rozšířené zobrazování záznamů a hledání v archivu je k dispozici hlavní menu *aktuality*.

4.2.1.2 Odkazy

Jako doplňkovou funkčnost je možné zadávat nejzajímavější odkazy na externí stránky nebo stránky univerzity. Rozšiřuje se tak síť relevantních webových stránek, které mohou být uživatelům lehce dostupné.

4.2.1.3 Vyhledávání

Vyhledávání bude teoreticky nejpoužívanější část aplikace. Pomocí možností, které aplikace nabízí lze vypisovat a nabízet uživateli užitečná data a s tím spojené grafické podklady. Jedná se o nejrychlejší způsob jak získat potřebné výsledky a obejít tak pomocné komponenty jako jsou plány budovy A se zakreslenými místnostmi.

4.2.1.4 Patra

Hledá-li uživatel konkrétní místnost, pak lze využít menu patra, kde podle podlaží může vybírat destinace a zpřístupňovat si animace a dodatečné informace. Samozřejmostí je zobrazování detailů o místnostech, kdo zde sídlí a podobně.

4.2.1.5 Místnosti

Zde se projevuje plná interaktivita prostředí Flash. Pro každé podlaží je k dispozici prostorový plán s označením místností a zobrazením detailů po najetí myší. Tyto data jsou načítána z databáze a lze je tedy kdykoli upravovat. Další funkčnost je zobrazení barevného rozlišení místností podle funkce a zobrazení stavebních plánů pro podrobné prohlížení.

4.3 Funkčnost aplikace

4.3.1 Obsah v rozhraní aplikace

Uživatel, který přistupuje k aplikaci vidí v podstatě pouze uživatelské rozhraní, které mu zprostředkovává interakci s daty a prvky. Při práci se mu tedy zobrazuje obsah v podobě výpisů dat, komponent, které zpřístupňují další informace, načítacím rozhraním, odvádějícím pozornost od časových prodlev vznikajících při načítání větších datových objemů a jiných.

Obsah je složen z mnoha prvků, které se dají rozdělit podle způsobu, jak jsou do prostředí zavedeny.

4.3.1.1 Generovaný obsah

Stále častějším způsoben vkládání ovládacích a informačních prvků je jejich generování skrze programový kód. Tato metoda je v actionskriptu dovedena k dokonalosti od verze 3.0, kdy uplatňují specifičtější metody objektového programování. Také nové komponenty dávají snadnější přístup k jednotlivých parametrům a tvorba obsahu programováním je pak zjednodušená. Mezi komponenty generované ryze dynamicky programově patří textová pole s obsahem z databáze nebo XML souborů, potom plochy s grafickými prvky statickými nebo i animovanými. AS k tomuto účelu poskytuje knihovny pro práci s gradienty, barvami, tvary a jiné.

4.3.1.2 Obsah z knihoven v prostředí Flash CS3

Stejně jako u jiných programovacích jazycích i ve Flashi jsou k dispozici knihovny prvků. V mém konkrétním případě jsem využil tyto základní prvky pouze u vyhledávání a to konkrétně combo boxy a check boxy. Tím spolupráce s definovanými komponentami skončila a na řadu přišly vlastní komponenty a knihovní prvky. Pro lepší představu však musím rozlišit mezi knihovnou, kterou používá Actionscript a knihovnou spolu s komponentami, kterou má k dispozici pracovní prostředí Flash.

V případě AS knihovny se jedná o systém programovaných tříd, které jsou uložené v adresářové struktuře prostředí Flash. Tyto kusy kódů jsou pak k dispozici skrze programovací jazyk a dávají nám možnost vytvářet i grafický obsah (generovaný grafický obsah).

Knihovny, které jsou k dispozici v prostředí Flash, jsou již hotové grafické prvky, kde není potřeba programově zasahovat do jejich vzhledu nebo je jinak definovat. Tyto prvky je možné vkládat rovnou na časovou osu nebo je přidávat také pomocí kódu (jsou-li správně předdefinované pro Actionscript) a *addchild* příkazu.

Konkrétně pro tuto aplikaci jde o kreslené plochy, předdefinovaná tlačítka s animacemi, plochy s designem importovaným z jiných kreslících nástrojů a předpřipravenými textovými plochami a podobně, které jsou již při exportování výsledného .swf souboru vloženy do vrstev nebo jsou vloženy při kompilaci skriptu. Například: Plocha pro výpis novinek po kliknutí na tlačítko menu *aktuality* je graficky poměrně složitá a její definice pomocí programování by byla velice obšírná. Proto jsem v prostředí Flash CS3 vytvořil kompletní grafický návrh pomocí kreslících nástrojů a ten pak následně zkonvertoval jako movieClip. Po odstranění tohoto kusu grafiky z časové osy mi zůstává jeho reference v knihovně, kde stačí tento prvek *exportovat pro actionscript* a nechat mu vytvořit automatickou definici třídy (například class: *plocha_vypis_mc*) při kompilaci. Během této kompilace se zpracuje i následující kód, který přidá komponentu na časovou osu.

```
var myVypis:MovieClip = new plocha_vypis_mc;
stage.addChild(myVypis);
setVypisObsah();

function setVypisObsah():void  { . . . }
```

SetVypisObsah() je volaná funkce, která naplní výpis dalšími daty či grafikou.

4.3.1.3 Načítaný obsah

Asi nejdůležitější obsah je ten, který je načítán za chodu aplikace. Nejzákladnější jsou data získávaná z databáze prostřednictvím PHP a dotazů s parametry. Tyto data jsou v drtivé většině posílána zpět do aplikace v podobě XML. Následně jsou generovaná textová pole, nastaveny jsou styly a formátování a poté zobrazeny uživateli. Podobně se pracuje s informacemi uloženými v souborech XML bez zásahu PHP. Konkrétně pak XLM s nastavením, nápovědou a dalšími méně důležitými daty, kde se očekává, že se nebudou aktualizovat tak často a proto není nutné je mít jako součást databáze. Blíže ukážu názorný příklad generování textů z databáze později v dalších kapitolách.

Další typ načítaných dat jsou obrázky. Většinou jsou ve formátu jpg, který dává dobrou kontrolu kvality, komprese a velikosti. Adobe Flash také velmi dobře podporuje formát png, ale ten je většinou součástí grafiky v knihovně, tudíž není načítán externě.

Funkčně důležité je také načítání plánů budovy A v externích souborech. Dává to větší flexibilitu v úpravách podkladů a také se zmenšuje velikost úvodní aplikace. Plány jsou tedy načteny jako externí soubory a zapracovány do aplikace tj. jsou jim přidány eventListenery, jsou vloženy do aplikace na své místo, popsány podle dat v databázi a oživeny tlačítky pro další akce.

Poslední a nejdůležitější externí soubory jsou průlety budovou. Celá aplikace je v podstatě konstruována tak, aby uživatele ve výsledku odeslala na animaci nějakého vybraného průletu budovou A. Zde jsou finální data, popisy a možnost pravé navigace. Tyto průlety jsou uloženy v adresářové struktuře odpovídající patrům budovy a každý soubor má unikátní jméno. Tímto se zajišťuje dynamická správa grafických podkladů např.: chcete udělat vtipný průlet k nějakému zaměstnanci pro nějakou událost? Jednoduše přepíšete stávající soubor s číslem místnosti a je hotovo.

4.3.2 Uživatelské rozhraní

V této kapitole se pokusím blíže nastínit jednotlivé prvky uživatelského rozhraní. Na

obr. 2 rozhraní aplikace

následujícím obrázku je kompletní rozhraní aplikace se zapnutými hlavními prvky.

Jak lze vidět, tak jsou zde pravá a levá plocha, centrální plocha, prostor pro nápovědu a místo kde se nachází menu.

4.3.2.1 Oblast menu

Menu je vymezeno tak, aby se zde mohlo umístit až 8 položek. Tyto položky jsou nastaveny přes XML soubor. V současné chvíli je zpřístupněno jenom šest položek a každá z nich aktivuje svoje vlastní procedury a používá jiné prvky v aplikaci.

Vytvoření menu je vázáno na XML soubor starající se o nastavení. Následující kód obsahuje stěžejní úryvek actionscriptu.

XML listy slouží k uchování těch elementů XML souboru, které se týkají dané oblasti.

```
var helpItems:XMLList = new XMLList();
var setupXML:XML;
var menuItems:XMLList = new XMLList();
var odkazyItems:XMLList = new XMLList();
var legendaColors:XMLList = new XMLList();
```

následuje vyžádání URL odkazu a jeho spuštění. Výsledkem je načtený XML soubor s daty pro nastavení aplikace.

```
var setupXMLrequest:URLRequest = new URLRequest("XML/cz.xml");
var setupXMLloader:URLLoader = new URLLoader(setupXMLrequest);
setupXMLloader.addEventListener(Event.COMPLETE, setupXMLloader_nacti);
```

pokud vše proběhne v pořádku, pak jsou naplněny XML listy a zavolány funkce pro jejich použití a zobrazení informací, které obsahují.

```
function setupXMLloader_nacti(e:Event):void {
      if(setupXMLloader.data){
            setupXML = XML(setupXMLloader.data);
            menuItems = setupXML.menu.menuItem;
            helpItems = setupXML.aplicationHelp;
            odkazyItems = setupXML.odkazy.odkaz;
            legendaColors = setupXML.legendaBarvy.barva;
            populateMenu();
            populateHelp();
            vytvor_uvitani();
      }
}
```

4.3.2.2 Centrální plocha

Je prostor pro hlavní datový obsah. Podle potřeby je zde vytvořeno rozhraní pro vyhledávání, výpis aktualit a zobrazení plánů místností. Jak přesně funguje zobrazování jednotlivých menu je mimo rámec této práce, a proto se mu zde nebudu věnovat.

4.3.2.3 Levá plocha a systém zobrazení

Za zajímavé řešení považuji levý panel, kde se zobrazují výsledky rychlého vyhledávání, odkazy a úvodní text. Vzhledem ke způsobu jak panel pracuje, je možné zde zobrazit jakékoli množství nezávislých textů.

Celý prvek funguje na principu rolovaní (za použití Tweener knihoven) a podle pořadí v jakém přicházejí požadavky na zobrazení textů se původní obsah posouvá níže. To samé funguje, když se obsah zhasíná a v obou případech, jak při vložení obsahu a jeho zhasnutí, se přepisují indexy jednotlivých obsahů. Této komponentě nezáleží ani na tom, o jaký typ dat se jedná. Může se zde tedy načítat textový obsah, obrázky, movieclipy, přehrávače a jiné podle potřeby. O efekt rolování a zobrazování obsahu se stará procedura, pracující s kontejnery pro data. Jediná limitace je, že nutné procedury lze spouštět skrze Actionscript a tedy zasahovat do kódu aplikace. Samozřejmě je možné naprogramovat nějaký systém, jak dát uživateli možnost linkovat data s tímto panelem bez znalosti programovacího jazyka, ale tato funkčnost zatím není nutná.

Následuje přehled kódu, který zajišťuje fungování levého panelu. Pokud chceme přidat do panelu odkazy, vytvoříme movieClip, který bude obsahovat, všechna data. Ten pojmenujeme a vzhledem k tomu, že odkazy jsou přidány jako poslední položka, nastavíme polohu v Y na 0.

```
var odkazy_mc:MovieClip = new MovieClip();
odkazy_mc.name = "odkazy_mc";
odkazy_mc.x = 0;
```

proměnná pro zjištění jestli jsou odkazy zapnuty

```
var odkazy_active:Boolean = false;
```

funkce odkazy_funkce je volána kliknutím na tlačítko menu „odkazy“

```
function odkazy_funkce():void {
      if(odkazy_active == false) {
            odkazy_mc.y = 0;
```

proměnná „posun" zajišťující nepřekrývání obsahu v movieClipu odkazy_mc

```
            var posun:int = 0;
```

odkazy_items je typu XMLList a je vytvořen při definici menu (výše), makeTextField je funkce, která vytváří textové pole s danými parametry.

```
            for (var i:int=0;i<odkazyItems.length();i++){
                  var polozka_mc:MovieClip = new MovieClip();
                  polozka_mc.x = 0;
                  polozka_mc.y = posun;
                  var ont:TextField = makeTextField (5, 5, 210, 50,
                  TFnormal2, odkazyItems[i].nazev, false, false, false,
                  false);
                  ont.height = ont.textHeight;
                  polozka_mc.addChild(ont);
                  var olt:TextField = makeTextField (5,ont.height, 210,
                  50, TFnormal4, odkazyItems[i].link, false, false, false,
                  false);
                  olt.height = olt.textHeight;
                  polozka_mc.addChild(olt);

                  var ot:TextField = makeTextField (5,ont.height +
                              olt.height, 210, 50, TFnormal4,
                              odkazyItems[i].popis, false, false, false,
                              false);
                  ot.height = ot.textHeight +5;

                  polozka_mc.addChild(ot);

                  posun += polozka_mc.height+10;
                  odkazy_mc.addChild(polozka_mc);
            }
```

Odkazy_close se definuje jako nová třída „levy_panel_content_close". Dále se mu nadefinují vlastnosti a eventListenery a přidá se do display listu daného movieClipu.

```
            var odkazy_close:levy_panel_content_close_mc = new
                                    levy_panel_content_close_mc();
            odkazy_close.x = 215;
            odkazy_close.y = odkazy_mc.height +10;
            odkazy_close.buttonMode= true;
            odkazy_close.close_txt.defaultTextFormat = TFnormal3;
            odkazy_close.close_txt.embedFonts = true;
            odkazy_close.close_txt.antiAliasType = AntiAliasType.ADVANCED;
            odkazy_close.close_txt.text = "zavrit odkazy";
            odkazy_close.addEventListener(MouseEvent.MOUSE_OVER,
                                    button_tam);
            odkazy_close.addEventListener(MouseEvent.MOUSE_OUT,
                                    button_zpet);
```

```
            odkazy_close.addEventListener(MouseEvent.CLICK,
                        odkazy_funkce_hide);
            odkazy_mc.addChild(odkazy_close);
```

následující podmínka a cyklus se starají o přesunutí stávajícího obsahu a uvolnění prostoru vkládanému movieClipu. Tweener class funguje tak, že transformuje jakoukoli číselnou hodnotu nějakého cíle na jinou za stanovený čas, a pak je možné deinovat, co se má provést po dokončení transformace „onComplete".

```
            if (this.levy_panel_mc.content_mc.y != 0) {
                  Tweener.addTween(this.levy_panel_mc.content_mc, {y:0,
                        time:0.5,transition:"easeOutQuart"});
                  this.levy_panel_mc.sb_mc.handle_mc.y = 0;
            }

            for(var j:int = 0; j <
            this.levy_panel_mc.content_mc.numChildren; j++) {
                  var cil:MovieClip =
                  this.levy_panel_mc.content_mc.getChildAt(j);
                  var puvodniY:int = cil.y;
                  Tweener.addTween(
                  this.levy_panel_mc.content_mc.getChildAt(j),
                  {y:puvodniY + l_panel_content_y1, time:0.5,
                  onComplete:odkazy_funkce_zobraz,
                  transition:"easeOutQuart"});
            }
      }
}
```

Funkce volaná po dokončení transformace definované v Tweener.

```
function odkazy_funkce_zobraz():void {
      odkazy_active = true;
      this.levy_panel_mc.content_mc.addChild(odkazy_mc);
}
```

Odkazy funkce hide využívá stejných principů jako předchozí funkce. Zde nejprve nadefinuje „target" movieClip, který bude zprostředkovávat daný movieClip, ten pomoci while cyklu vyčistíme, odstraníme z display listu a přesuneme zbývající obsah v levém panelu místo něho.

```
function odkazy_funkce_hide(e:MouseEvent):void {
      var target:MovieClip =

      this.levy_panel_mc.content_mc.getChildByName("odkazy_mc");
            var vyska:int = target.height;
            while(target.numChildren > 0 ) {
                  target.removeChildAt(0);
            }
      this.levy_panel_mc.content_mc.removeChild(odkazy_mc);
      Tweener.addTween(this.levy_panel_mc.content_mc, {y:0,
      time:0.5,transition:"easeOutQuart"});
```

Cyklus pro přesun kusů zbývajícího obsahu na místo odstraněného movieClipu.

```
      for(var j:int = 0; j < this.levy_panel_mc.content_mc.numChildren;
      j++) {
            var cil:MovieClip=this.levy_panel_mc.content_mc.getChildAt(j);
```

```
            var puvodniY:int = cil.y;
            if (puvodniY != 0) {
            Tweener.addTween(this.levy_panel_mc.content_mc.getChildAt(j),
            {y:puvodniY - vyska, time:0.5,
            onComplete:odkazy_funkce_change_active,
            transition:"easeOutQuart"});
            }
      }
      l_panel_content_y1 = 0;
}
```

Nakonec nastavíme obsah odkazů na false, aby bylo možné jej znovu zobrazit v případě potřeby.

```
function odkazy_funkce_change_active():void {
      this.levy_panel_mc.sb_mc.handle_mc.y = 0;
      odkazy_active = false;
}
```

4.3.3 Systém programování aplikace

4.3.3.1 Programování na základě tříd a procedurální programování

V programovacím jazyku Actionscript lze vytvářet kód hned několika způsoby. V nové verzi 3.0 je nejčastěji používaný přístup skrze třídy. Nejlepší je uvažovat o třídách jako o reálných objektech. Můžeme třeba definovat třídu "auto", které má nějakou barvu, výkon, používá určitý typ paliva a podobně. Podobně můžeme vytvořit třídu pro jiné objekty a v tomto případě se jedná o objekty spojené s funkčností aplikace. Takový objekt může být třeba tlačítko, které má nějaký tvar, popis a barvu. V tomto případě definujeme tedy "class Tlacitko" . V této třídě můžeme přistupovat k jejím dalším vlastnostem. Třídy mohou tedy reprezentovat jak fyzické objekty, tak i objekty abstraktní, které nám poskytují například pouze nějaké doplňující informace k tématu nebo o jiném objektu.

Třídy obsahují informace a jiné operace. Narozdíl od procedurálního programování, dávají třídy našim datům větší pořádek a přehled. Máme tak skupiny objektů, vlastností a akcí, které jsou jednoznačně spjaté s daným objektem.

Řekněme že tvoříme složitý program, který se má starat o nákup zboží, zásobování, vybírání poplatků a jiné, pak vytvoříme skupinu tříd, kde se každá stará o danou část. Máme

tak daný systém, ve kterém můžeme odděleně upravovat jednotlivé funkce. Pro jednu třídu můžeme definovat hned několik jejich instancí, které jsou všechny typem třídy (např.: class Tlacitko).

Principy, na kterých dále třídy pracují, jsou mimo rámec této práce a lze je najít v každé publikaci o některém programovacím jazyku.

V rámci navigace v budově A je většina kódu je zpracována jako procedurální. Kód je vložen na jednom snímku v časové ose čímž se zvyšuje přehlednost. Ovšem s rostoucím počtem řádků scriptu se i zmenšuje čitelnost pro nezainteresovaného pozorovatele nehledě na to, že v aplikaci je více než 3000 řádků programu.

Jednou z věcí, které jsou malou nevýhodou při vytváření aplikací ve flash authoring tool jako je Flash CS3, je tendence uživatele aplikovat kód přímo na časovou osu, i když by bylo mnohdy lepší použít class based programování. Možnost aplikovat scripty přímo na komponenty velice zlehčuje práci, a proto se mnohdy programátor uchýlí k jednoduššímu přístupu před upřednostněním větší přehlednosti. Volba stylu programování vychází z použití animací, kde je časová osa přirozenou vlastností.

4.3.3.2 Generování komponent

Většina prvků, které se objevují v aplikaci jsou programově generované na základě obsahu, který se získává z databáze. Po získání dat se používají definované funkce, které se postarají o vypsání obsahu. Postup jak se načítají data v AS 3.0 je popsán níže. Samozřejmě některé komponenty jsou již předem vytvořené a pouze se přidávají do display listu a vloží se do nich data. To je nejjednodušší proces jak vypsat uživateli informace. Tento postup s sebou nese nevýhodu a to především ve větší velikost aplikace při načítání ve webovém prohlížeči. Vytváření komponent za běhu je méně náročné na datovou velikost, ale na druhou stranu se používá větší množství operační paměti, když je těchto komponent více najednou. Následuje příklad funkční procedury, která vytváří textová pole s vyhlazeným písmem, fonty připravenými pro animaci a deformace.

Vytvoříme embeded (zapečený) font, který je potom možné animovat a deformovat aniž by text zmizel nebo se zobrazoval chybně

```
var tahomaFont:Font = new TahomaEmbed();

TextFormat fomrátuje text v textovém poli podle jeho definice

var TFnormal2:TextFormat = new TextFormat();
TFnormal2.font=tahomaFont.fontName;
TFnormal2.size = 11;
TFnormal2.color = 0x333333;

chceme-li vytvořit textové pole s daným textem, vytvoříme si proměnnou do
které textové pole uložíme. Ideální je se stejným datovým typem, abychom
se vyhnuli chybám způsobeným růzností datových typů.

var textovePole:TextField = makeTextField(0, 0, 100, 50, TFnormal2,
"ahoj",     false, false, false, false);

function makeTextField(mojeX:int, mojeY:int, mojeW:int, mojeH:int,
      format:TextFormat, obsah:String, mojeMultiline:Boolean,
                        mojeWordWrap:Boolean, mojeSelectable:Boolean,

      mojeMouseEnabled:Boolean) :TextField {

      var mojeTF:TextField = new TextField();

      mojeTF.defaultTextFormat = format;
      mojeTF.embedFonts = true;
      mojeTF.antiAliasType = AntiAliasType.ADVANCED;
      mojeTF.multiline= mojeMultiline;
      mojeTF.wordWrap= mojeWordWrap;
      mojeTF.selectable= mojeSelectable;
      mojeTF.mouseEnabled = mojeMouseEnabled;
      mojeTF.htmlText = obsah;
      mojeTF.width = mojeW;
      mojeTF.height = mojeTF.textHeight + 5;
      mojeTF.x = mojeX;
      mojeTF.y = mojeY;

      return(mojeTF);
}
```

Po provedení tohoto kódu se nám vrátí textové pole se zobrazeným popisem v daném textovém formátu.

4.3.3.3 Načítání obsahu

Načítání obsahu z databáze nebo webového serveru je jedním z nejdůležitějších postupů použitých v celé aplikaci, jelikož je celá založená na datech z databáze. K tomu je v AS 3.0 zapotřebí použít hned několik tříd, narozdíl od předešlých verzí Actionscriptu, kde

se načítání dat z daného odkazu dalo provést zápisem dvou řádků. V AS 3.0 je postup, uvedený v obecném příkladu. Tento příklad je funkční kus kódu.

```
nadefinujeme si string s cestou k souboru

var odkaz:String = "http://www.tul.cz/.../zpracujData.php";

hodnota zadaná v proměnné "poslanyText" se předá objektu URLVariables jako
hodnota přiřazená jako "poslanyTextVar". Dále se vytvoří URLRequest
objekt, který zpracuje odkaz. Nastavme metodu, kterou pošleme data (get
nebo post) a přiřadíme URLVariables (url proměnné). Jako poslední
vytvoříme objekt URLLoader, který se postará o poslání dat a získání
odpovědi. Přiřadíme mu událost "complete", a zpracujeme funkcí výsledná
data.

var poslanyText:String = "text";
var url_variables:URLVariables = new URLVariables();
url_variables.poslanyTextVar = poslanyText;

var urlRequest:URLRequest = new URLRequest(novinkyFile);
urlRequest.method = URLRequestMethod.POST;
urlRequest.data = url_variables;

var urlLoader:URLLoader = new URLLoader();
urlLoader.load(urlRequest);
urlLoader.addEventListener(Event.COMPLETE, data_handler);

function data_handler(e:Event):void {
      ...zde se zpracují přijatá data...
}
```

Stejně pracujeme i s XML soubory. Pouze zvolíme cestu k souboru a nemusíme se starat o url proměnné.

4.4 Aplikace v kódu

V této části je uveden kus AS 3.0 kódu pro případné vyzkoušení. Takto vypadá v aplikaci a pro jiné užití bude potřeba script upravit vzhledek názvu použitých komponent.

4.4.1 Interaktivní plány pater „místnosti“

Plány jednotlivých pater jsou zpracovány kombinací standardních knihoven AS 3.0 a Papervision3D knihoven. Pro zobrazení obsahu se provede několik na sebe navazujících činností, které dají dohromady samotný plán s texty, legendou a detaily v textových polích.

Podkladové animace (movie clipy) jsou předem vytvořené tak, aby obsahovaly interaktivní obsah, který je možné poté ovládat skrze vlastní aplikaci nalinkováním přes strukturu display objektů. Ty jsou uložené na daném místě v adresářové struktuře na serveru a jsou volány přes programový kód aplikace.

Pro kompletní plány jsou potřeba i textová data obsažená v databázi. Proto je v rámci aplikace volána funkce, která získá data prostřednictví PHP souboru k tomu určenému. V následujících řádcích je uveden kompletní programový kód zajišťující uživateli interakci s plány pater.

4.4.1.1 Actionscript 3.0 a Papervision3D plány

Následující kód činí kompletní zpracování dat a podkladových movie clipů. Jde o funkční příklad jak pracovat s externími daty v podobě databáze a předkompilovaných .swf souborů. V tomto textu budou uvedeny pouze důležité části kódu.

Nejprve si nadefinuji globální proměnné potřebné v rámci interaktivních plánů. Jsou to různé string proměnné obsahující cesty k souborům a pomocné hodnoty určující v tomto případě o jaké patro se jedná.

```
var MIST_file:String = "SWF/plan/PLAN_patro_1.swf";
var MIST_php_file:String = "PHP/mistnosti_plany.php";
var MIST_typy_file:String = "PHP/mistnosti_typy.php";
var MIST_nacitanePatro:String = "1";
```

Dále následují definice pro papervision3d a definování komponent použitých pro zobrazení obsahu.

```
var MIST_Loader:Loader;
var MIST_URLLoader:URLLoader;
var MIST_plan_mc:MovieClip;
var MIST_plocha:MovieClip;
var mistnosti_active:Boolean = false;
var MIST_viewport:Viewport3D;
var MIST_scene:Scene3D;
var MIST_camera:Camera3D;
var MIST_material: MovieMaterial;
var MIST_plane:Plane;
var MIST_renderer:BasicRenderEngine;
var planMaterialLink:MovieClip;
var MIST_barvy_active:Boolean = true;
```

Následují jednotlivé funkce, které se starají o načtení a zobrazení potřebných informací a grafiky. První funkce je volána stisknutím tlačítka menu místnosti. Všechny ostatní jsou sled událostí, které nastávají po zavolání první funkce. V první části funkce ošetřuji, jestli je zobrazený výpis novinek, který by mohl blokovat prostor, kde se má zobrazit plán místností.

```
function mistnosti_funkce():void {
      if (this.obsah_vypis_mc.x == 255)
      {
            aktuality_funkce_active = false;
```

Tento kus skriptu provede smyčku, která odstraňuje všechny display objekty obsažené ve vnořeném movie clipu"content.mc" rodiče "obsah_vypis_mc".

```
            while(this.obsah_vypis_mc.content_mc.numChildren > 0 ) {
                  this.obsah_vypis_mc.content_mc.removeChildAt(0);
            }
            while(this.obsah_vypis_mc.button_holder_mc.numChildren > 0 ) {
            this.obsah_vypis_mc.button_holder_mc.removeChildAt(0);
            }
      this.obsah_vypis_mc.x = -800;
      }
```

Otestujeme jestli v contejneru pro grafiku týkající se obsahu není žádný objekt a vyčistíme container pro zobrazení novinek „novinky_holder_mc".

```
      if (this.obsah_holder_mc.numChildren == 0) {
            while (this.novinky_holder_mc.numChildren > 0) {
            this.novinky_holder_mc.removeChildAt(0);
      }
```

Do předdefinované proměnné MIST_plocha vložíme definici třídy připravené podkladové grafiky.Nadefinujeme vlastnosti a polohu a vložíme ji grafiku addchild do kontejneru pro obsah.Také zavoláme funkce pro nastavení dalších komponent a načtení obsahu.

```
      MIST_plocha = new MIST_plocha_mc;
      MIST_plocha.MIST_plocha_close_mc.buttonMode = true;
      MIST_plocha.x = 255;
      MIST_plocha.y = 100;
      this.obsah_holder_mc.addChild(MIST_plocha);
      mistnosti_active = true;
      MIST_set_patra_buttons();
      MIST_load_content();
      }
```

Pokud se v konteineru nachizí již dříve spuštěný obsah, pak se provedou akce v následující podmínce.

```
      if(this.obsah_holder_mc.numChildren > 0 && mistnosti_active ==
false){
            while(this.obsah_holder_mc.numChildren > 0 ) {
            this.obsah_holder_mc.removeChildAt(0);
            }
            while (this.novinky_holder_mc.numChildren > 0) {
                  this.novinky_holder_mc.removeChildAt(0);
            }
            mistnosti_active = true;
            vyhledavani_active = false;
            patra_active = false;
            MIST_plocha = new MIST_plocha_mc;
            MIST_plocha.MIST_plocha_close_mc.buttonMode = true;
```

```
            MIST_plocha.MIST_plocha_close_mc.addEventListener(
                              MouseEvent.CLICK,hide_MIST_plocha);
            MIST_plocha.x = 255;
            MIST_plocha.y = 100;
            this.obsah_holder_mc.addChild(MIST_plocha);
```

Nastavíme uživatelské rozhraní a načteme obsah zavolání dvou dalších funkcí.

```
            MIST_set_patra_buttons();
            MIST_load_content();
      }
}
```

Následující funkce nastaví tlačítka v rozhraní plánů místností. Ty jsou použity pro přepínání mezi patry a barevným a stavebním zobrazením plánů. Po provedení nastavení se volá *MIST_load_legendu* funkce.

```
function MIST_set_patra_buttons():void {...}
function MIST_load_legendu():void {
```

vyčistíme grafický obsah uvnitř MIST_plocha display objektu

```
      while(MIST_plocha.MIST_legenda_mc.content_mc.numChildren > 0) {
            MIST_plocha.MIST_legenda_mc.content_mc.removeChildAt(0);
      }
```

Proměnná URLRequest a URLLoader jsou základní stavební kameny pro práci s odkazy na externá data. URLLoader načítá data a detekuje událost COMLETE, která říká, že data jsou načtena. Následně spouští handler načteného obsahu.

```
      var urlRequest:URLRequest = new URLRequest(MIST_typy_file);
      var urlLoader:URLLoader = new URLLoader();
      urlLoader.addEventListener(Event.COMPLETE,MIST_legenda_handler);
      urlLoader.load(urlRequest);

      function MIST_legenda_handler(e:Event):void {
            ...
      }
}
```

Funkce MIST_load_new_plan je v podstatě stejná jako úplně úvodní funkce s tím rozdílem, že zde se neinicializuje vytvoření podkladové plochy, nenastavuje se uživatelské rozhraní, ale spustí se znovu načítání externího obsahu tentokrát s jinými hodnotami. Ve funkcích, které se volají následovně, je ošetřeno odstranění původního obsahu.

```
function MIST_load_new_plan(e:MouseEvent):void {...}
```

V dalším kódu se postráme o načítání obsahu plánů, konkrétně externích .swf souborů.

```
function MIST_load_content():void { MIST_set_active_button();
      var urlRequest:URLRequest = new URLRequest(MIST_file);
      var loader:Loader = new Loader();
      loader.load(urlRequest);
      loader.contentLoaderInfo.addEventListener
            (ProgressEvent.PROGRESS,MIST_loading_progress);
      loader.contentLoaderInfo.addEventListener
            (Event.COMPLETE,MIST_clear_obsah);
      loader.load(urlRequest);
}
```

Funkce MIST_loading_progress a MIST_clear_obsah jsou ošetřením událostí načítání souborů.swf ze serveru. První z nich zobrazuje animaci doprovázející načítání a druhá se stará o vyklizení původního obsahu a zavolání funkcí pro načtení popisků k místnostem a zobrazením celého plánu v aplikaci.

```
function MIST_loading_progress(e:ProgressEvent):void {...}
function MIST_clear_obsah(e:Event):void {...}
function MIST_load_popisky():void {...}
```

Následující funkce MIST_handler je nejdůležitější z toho hlediska, že se zde formátuji data, která nám poskytnou načítací funkce skrze PHP a databázi. Složitost a rozsah skriptu je vyšší, proto je uveden v příloze.

```
function MIST_handler(e:Event):void {...}
function MIST_barvy_on_off(e:MouseEvent):void {...}
```

Funkce na dalších řádcích jsou pomocné, a starají se o události myši, tedy změnu barev po najetí myší a renderování prostorového výstupu z knihoven papervision3D.

```
function MIST_set_active_button():void {...}
function MIST_colorTransformTam(e:MouseEvent):void {...}
function MIST_colorTransformZpet(e:MouseEvent):void {...}
function MIST_render(e:Event):void {...}
```

Poslední z diskutované skupiny funkcí je funkce, která se stará o vyčistění obsahu komponent podílejících se na funkčnosti rozhraní plánů pater.

```
function hide_MIST_plocha(e:MouseEvent):void {
```

```
        stage.removeEventListener(Event.ENTER_FRAME, MIST_render);
        while (this.obsah_holder_mc.numChildren > 0) {
              this.obsah_holder_mc.removeChildAt(0);
        }
        MIST_plocha.MIST_plan_holder_mc.removeChild(MIST_viewport);
        MIST_scene.removeChild(MIST_plane);
        patra_active = false;
        vyhledavani_active = false;
        mistnosti_active = false;
```

loadNovinky je zde kvůli tomu, že rozhraní plánů si vyžaduje i prostor původně určený novinkám. Proto po odstranění obsahu plánů místností se znovu načítají boxy tří nejaktuálnějších novinek.

```
        loadNovinky();
}
```

Závěr

Aplikace je postavena na významných informačních technologiích dnešní doby, kterými jsou Adobe Flash, Autodesk Maya, PHP, MySQL databáze, programovací jazyk Actionscript 3.0. Design uživatelského rozhraní je kombinací interaktivních prvků, např: trojrozměrné panorama a aktivní plány pater, a prvků pro zobrazování dat z databáze a načítání dynamického grafického obsahu. Uživateli se nabízí několik způsobů získávání informací. Práce představuje navigační webový systém pro orientaci v hlavní poměrně rozsáhle budově Fakulty mechtroniky, informatiky a mezioborových studií.

Postupně jsou pojednány zásady a skutečnosti, které dávají umělým 3D objektům skoro dokonalou iluzi známou z reálných scenérií.

Datová část je postavená na serveru MySQL a v aplikaci nechybí webové rozhraní pro správu dat. Data, uložena podle popsaného datového modelu, jsou aktuální ke dni odevzdání této práce. Číselníky označení místností jsou reálná data tak, jak je používá správa budov TUL. Obsazení místností se může poměrně rychle měnit, proto je do budoucna potřeba, v případě praktického použití aplikace na webu fakulty, najít vhodného zaměstnance pro administraci těchto dat.

Textová část diplomové práce představuje přehled toho, jak je možné pomocí moderních a v současnosti relativně nových technologií vytvořit komplexní interaktivní systém a jak do něj principielně komponovat nové individuální postupy. I když je zde uvedeno několik praktických konkrétních postupů tvorby komponent a slučování obsahu do podoby funkční aplikace, je třeba výsledek brát jako jedeno z možných řešení a ne jako jediný přístup k podobné tématice.

Komplexní informační systém pro navigaci v budově A Fakulty mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci svou grafickou podobou může oživit webové stránky fakulty a být inspirací pro další informační systémy spojené 3D prostorem.

Seznam použité literatury a zdrojů

[1] Žára, Jiří. Moderní počítačová grafika. Computer Press, 1998.

[2] Moock, Colin. Essentials Actionscript 3.0. O´Reilly Media, 2007.

[3] Braunstein, Roger. Actionscript™ 3.0 Bible. Wiley Publishing, inc., 2008.

[4] Keefe, Matthew. Flash® and PHP Bible. Wiley Publishing, inc., 2008.

[5] http://www.root.cz

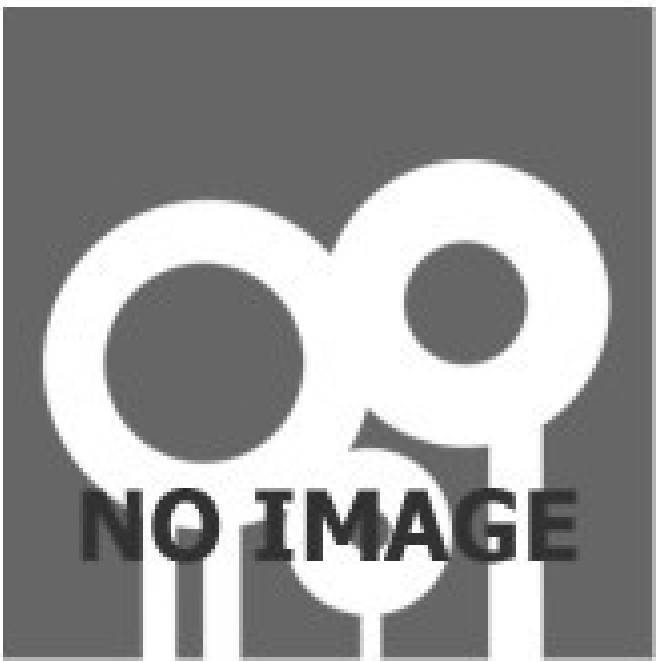
[6] http://www.thegnomonworkshop.com

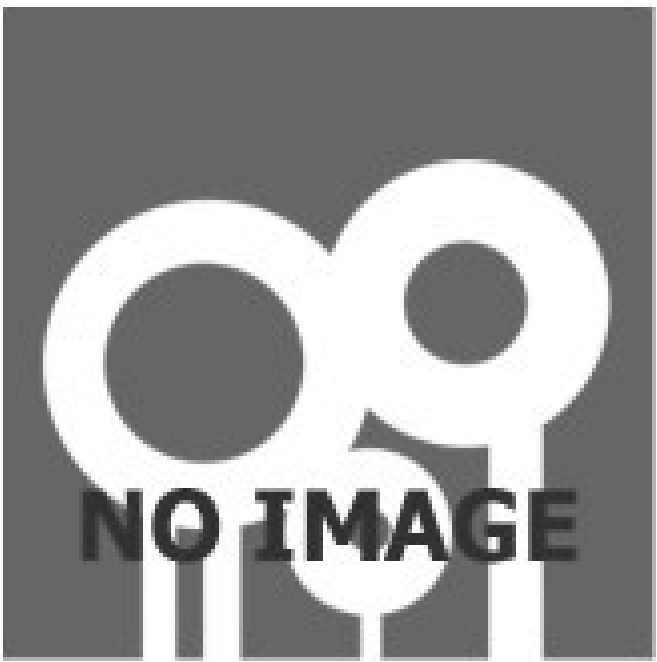
[7] http://www.digitaltutors.com

[8] http://www.wikipedia.org

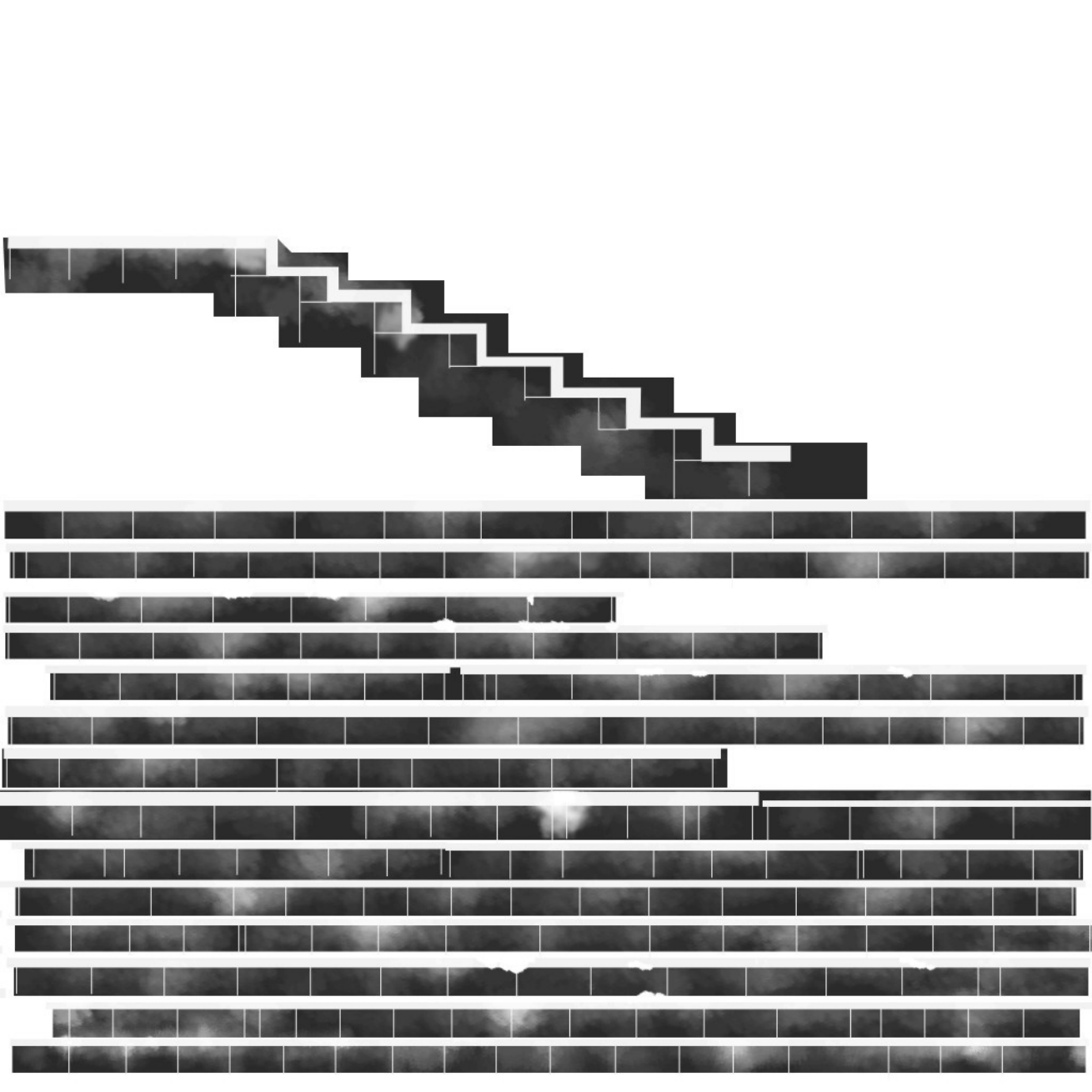
[9] http://www.adobe.com

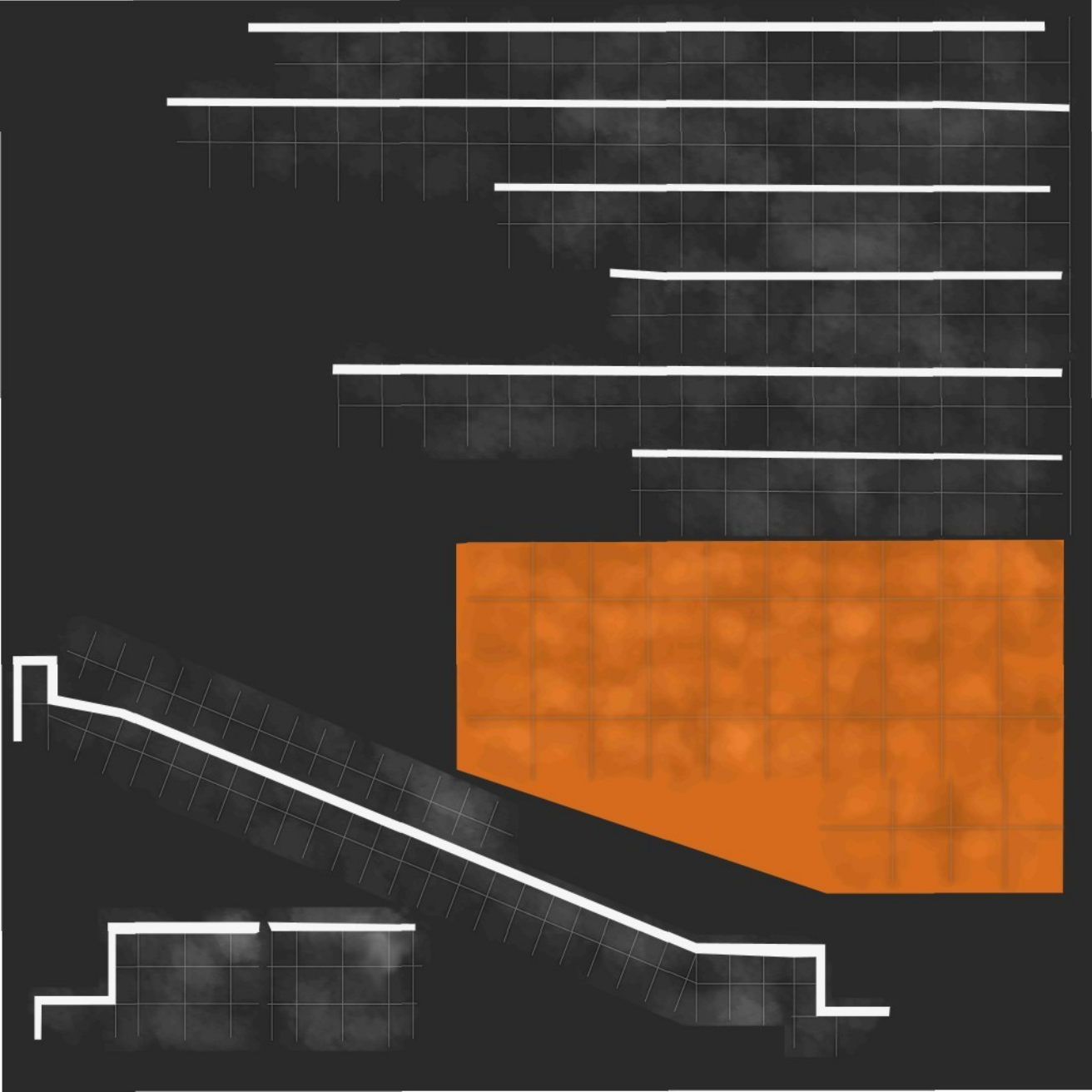
ADMINISTRACE DATABÁZE BUDOVY A

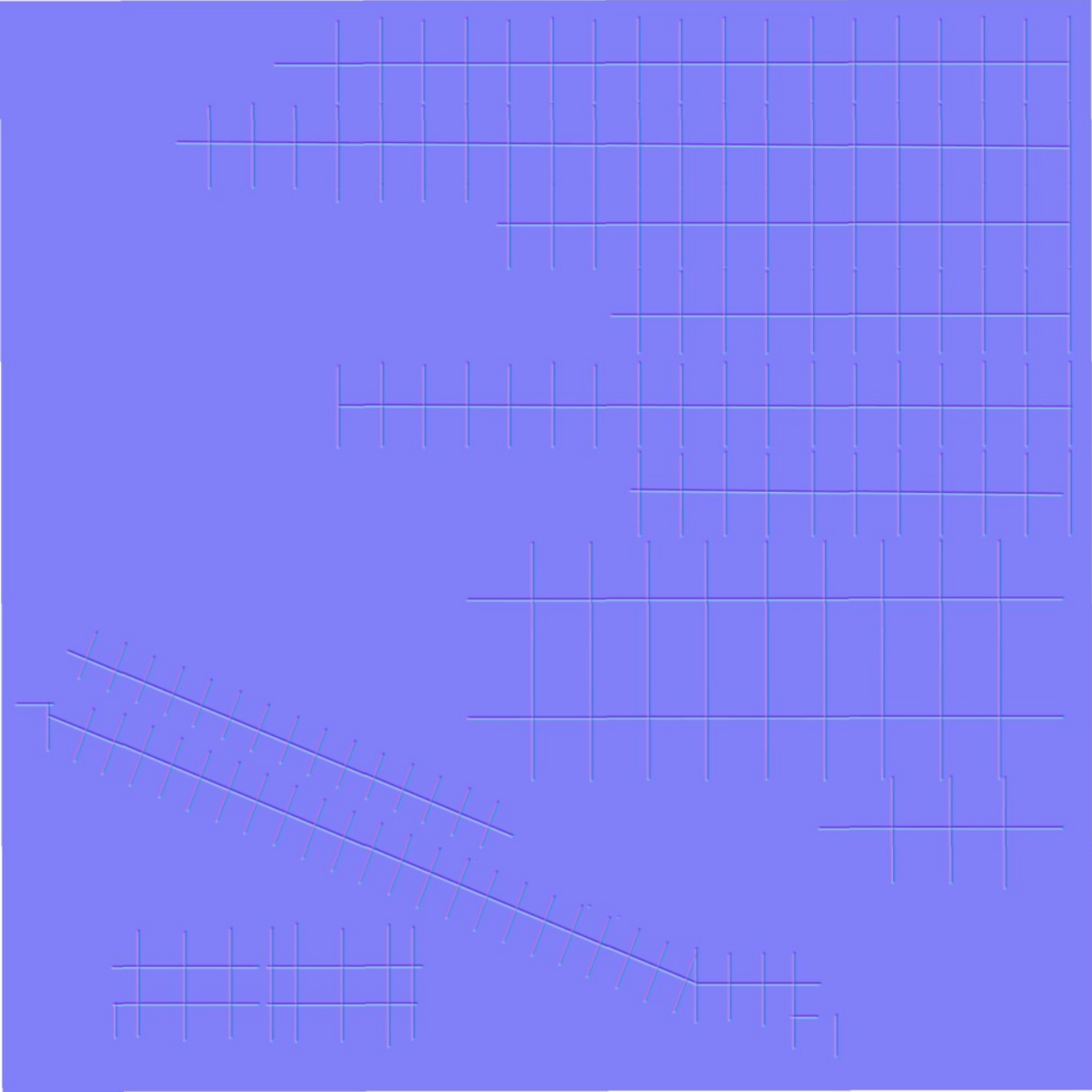

NO IMAGE

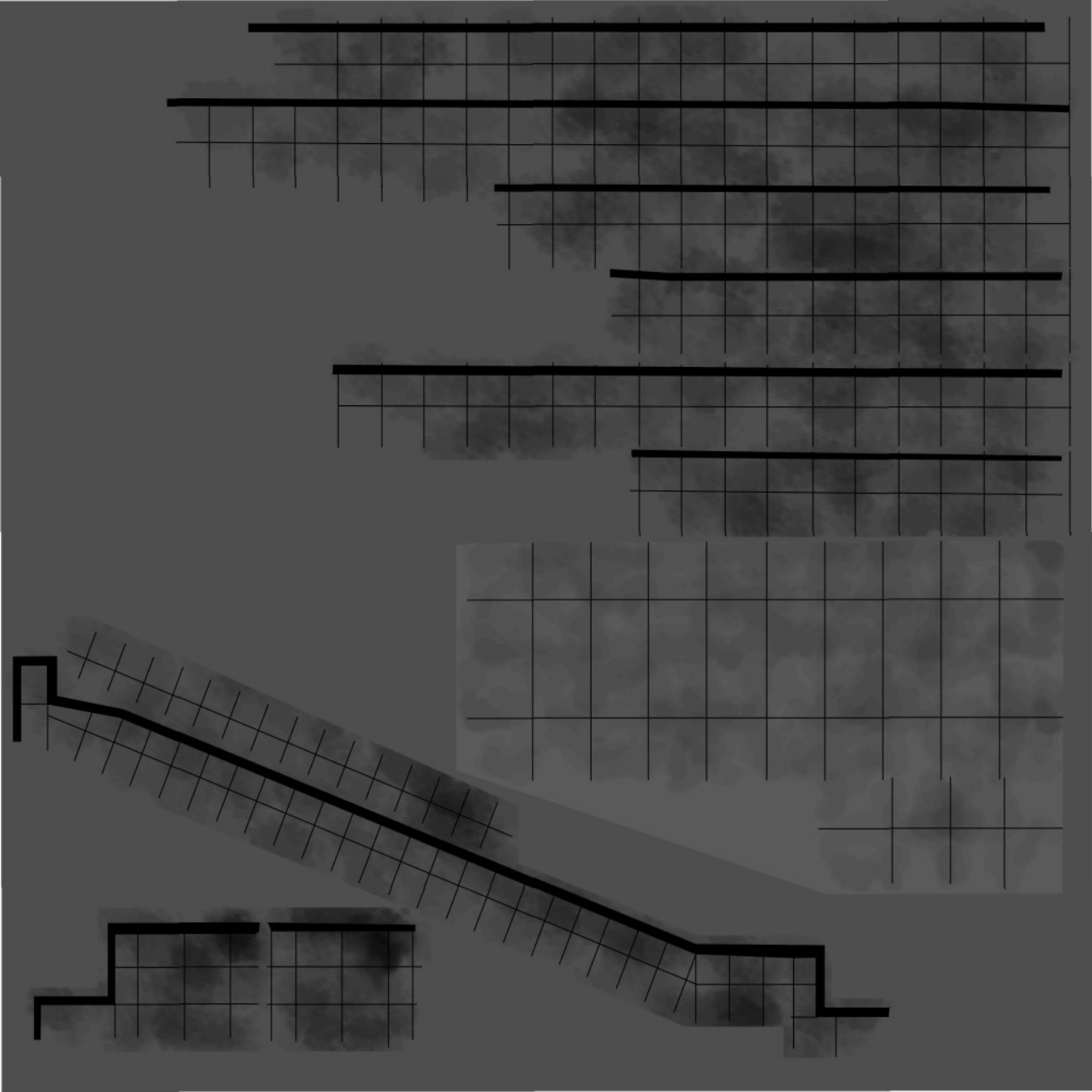

NO IMAGE

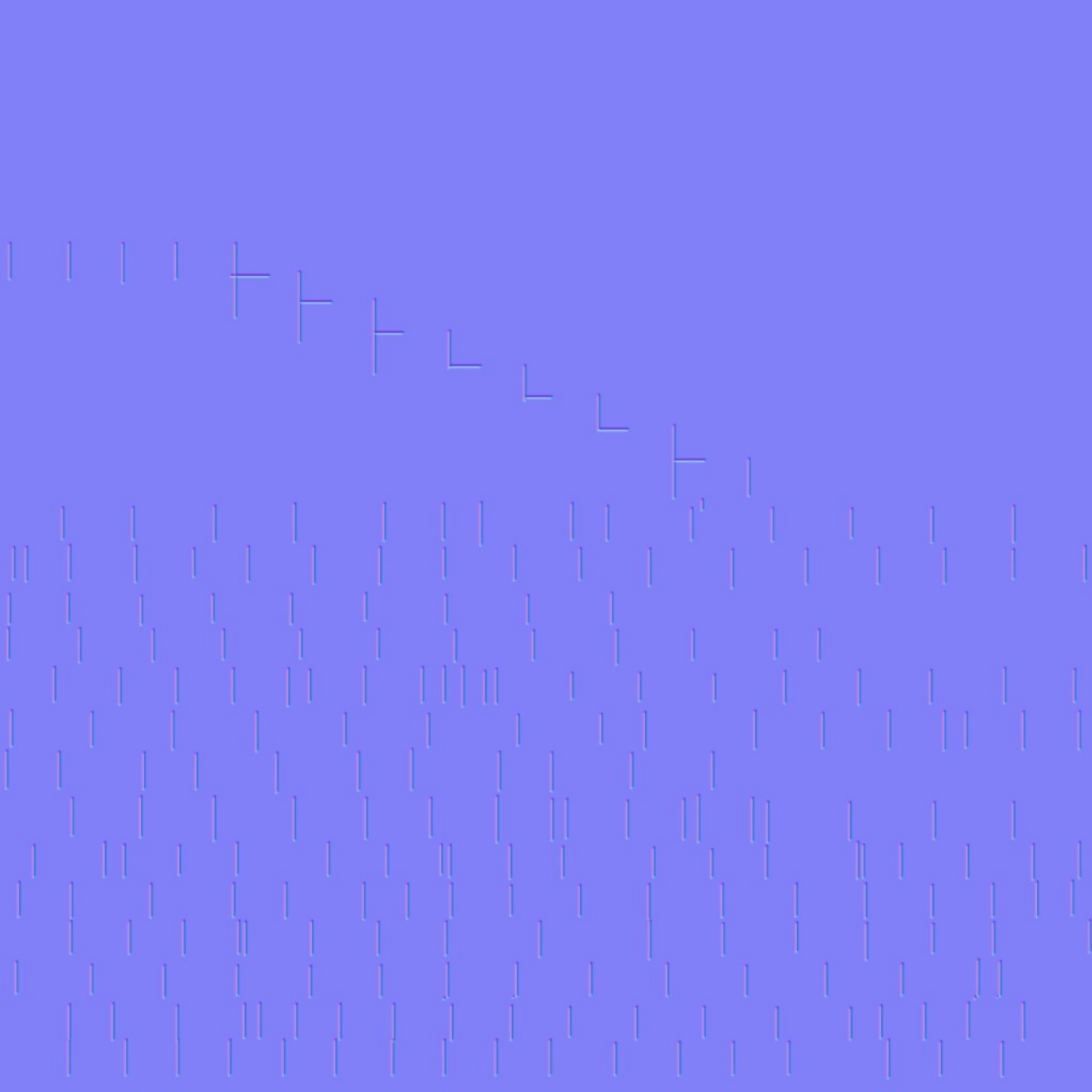
4. PATRO	ARCHITEKTONICKÁ KANCELÁŘ FAK. ARCHITEKTURY
3. PATRO	FM, NTI, RSS - KANCELÁŘE
	VC ARTEC, LIANE
2. PATRO	UČEBNY A3, AT3, A4, A10, A11
	FM, NTI - KANCELÁŘE, SEKRETARIÁT, VC ARTEC
1. PATRO	UČEBNY A1, A2, ZASEDACÍ MÍSTNOST
	DĚKANÁT FS, STUDIJNÍ ODDĚLENÍ FS
	DĚKANÁT FM, STUDIJNÍ ODDĚLENÍ FM
	FM, NTI - KANCELÁŘE, MTI - KANCELÁŘE
	VÝZKUMNÉ CENTRUM ARTEC
PŘÍZEMÍ	FM, MTI - KANCELÁŘE, UČEBNY A0, AP12
	LABORATOŘE AP9,AP11,TK3, TK4, TK6, TK7, TK8
SUTERÉN	LABORATOŘE - S14, S15, TK0, VC ARTEC
	FM, ITE - PCB AB.- VÝROBA A OSAZOVÁNÍ DPS
POZNÁMKA	ITE - ÚS AV INF. TECHNOLOGIÍ A ELEKTRONIKY
	MTI - ÚSTAV MECHATRONIKY A TECH. INFORMATIKY
	NTI - ÚSTAV NOVÝCH TECHN. A APLIK. INFORMATIKY
	RSS - ÚSTAV ŘÍZENÍ SYSTÉMŮ A SPOLEHLIVOSTI

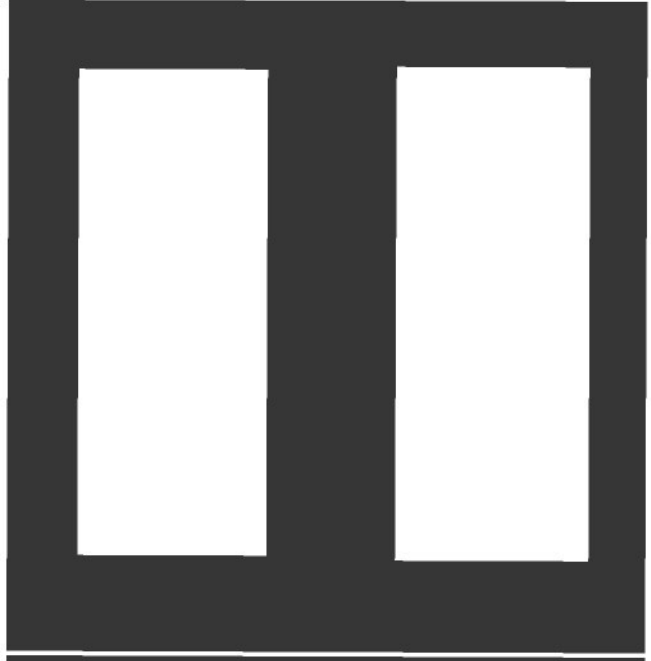


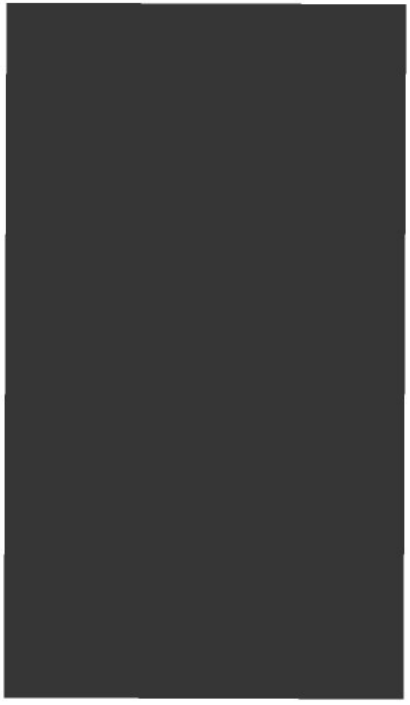
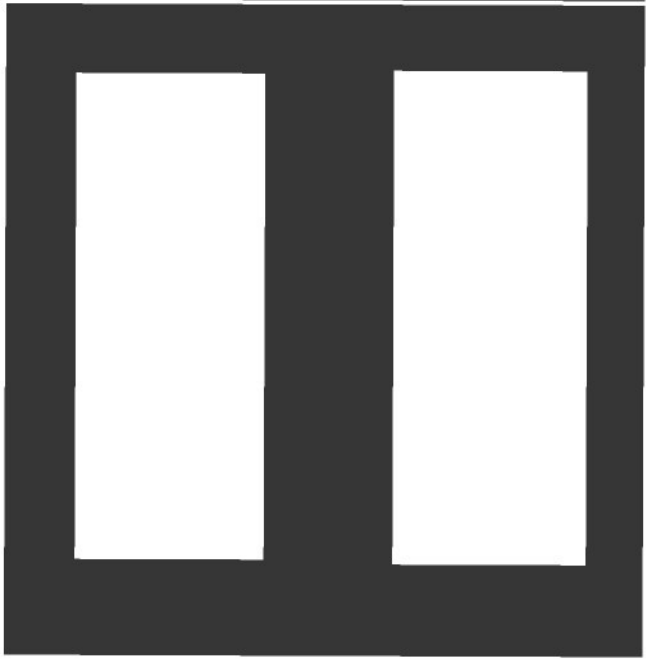
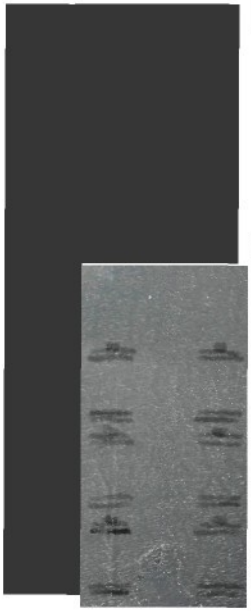


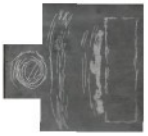




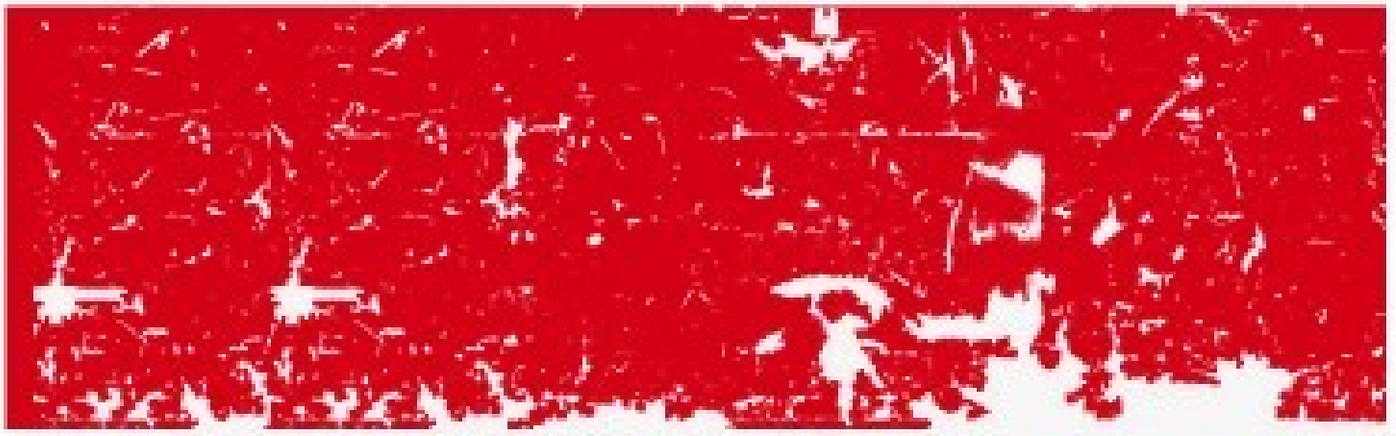


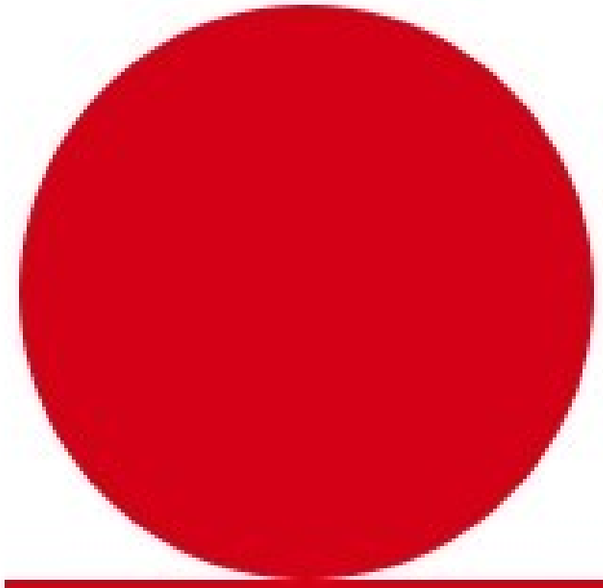


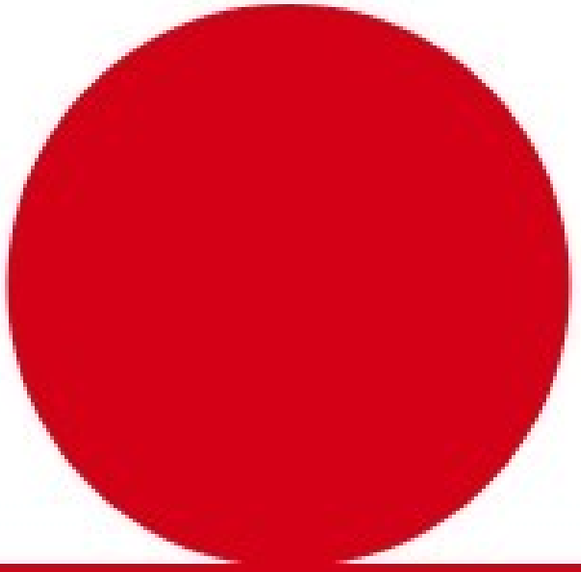





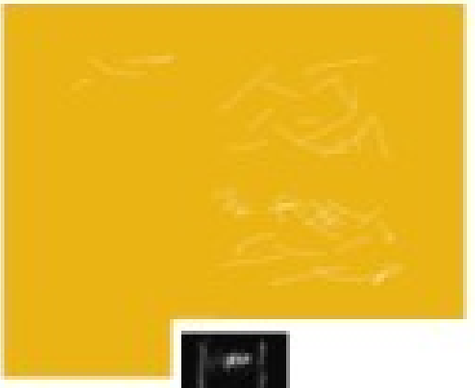


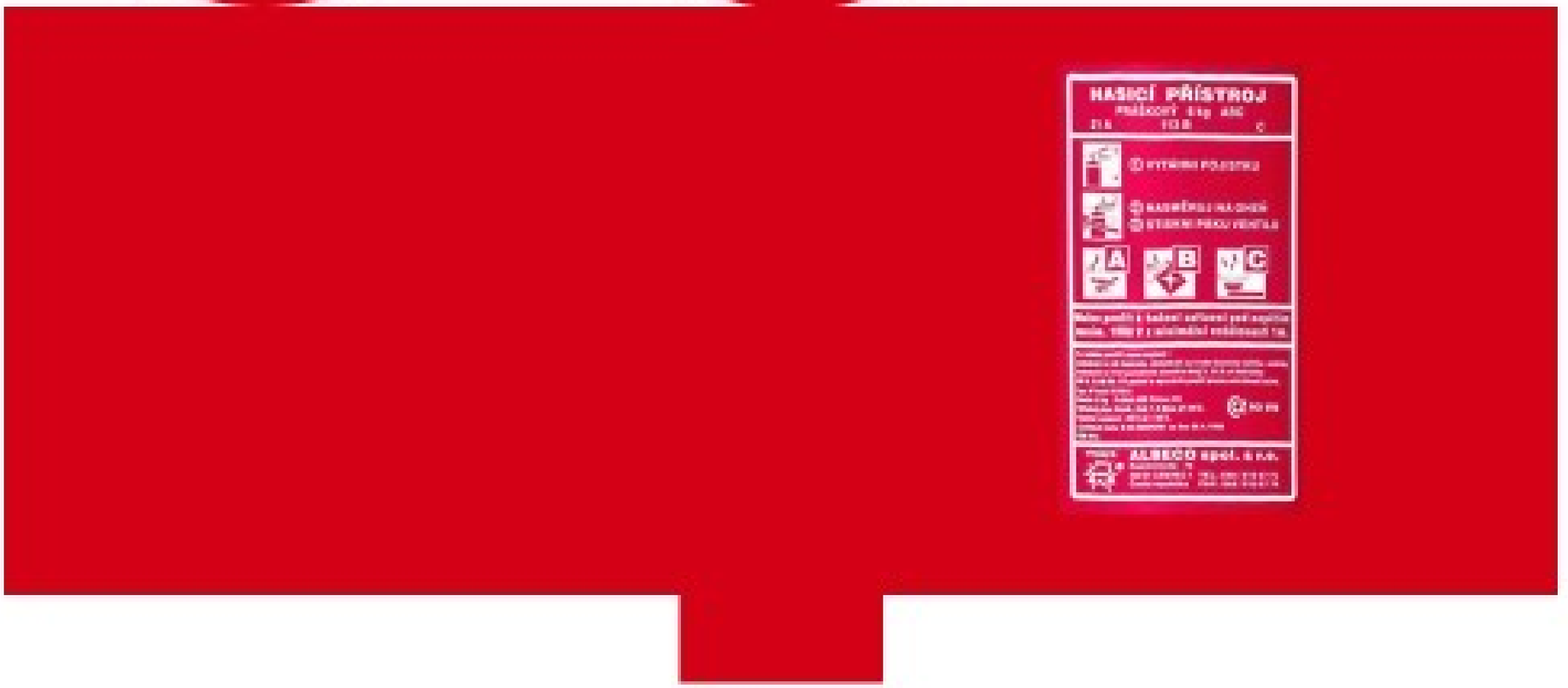


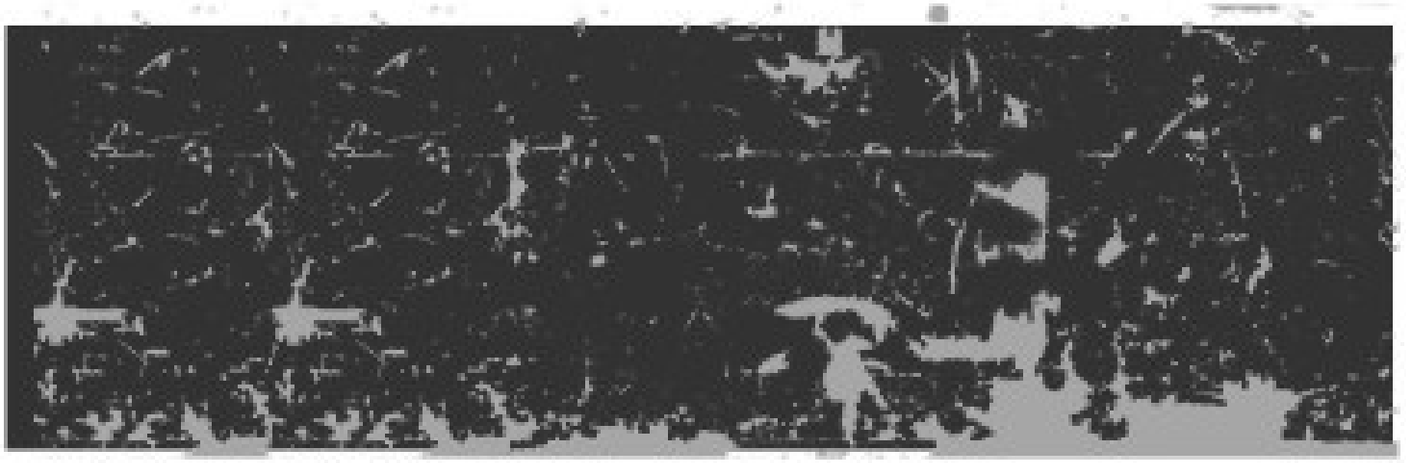








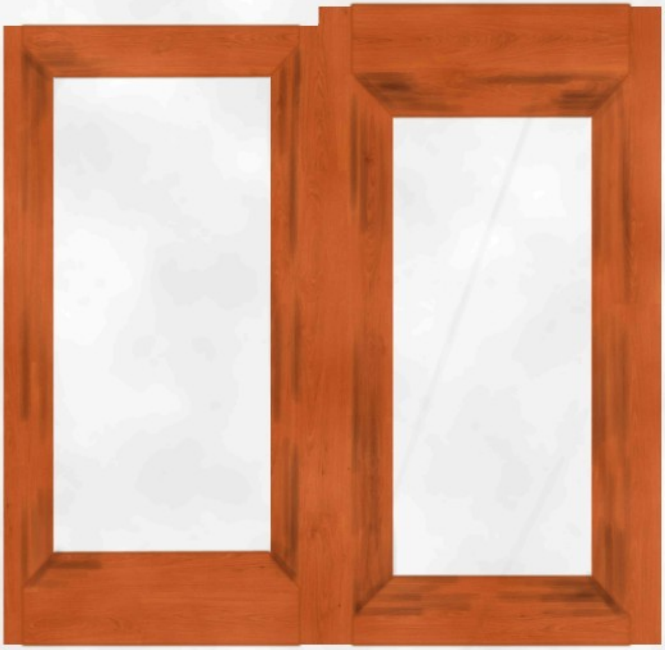

HASICÍ PŘÍSTROJ
ALBECO spol. s r.o.



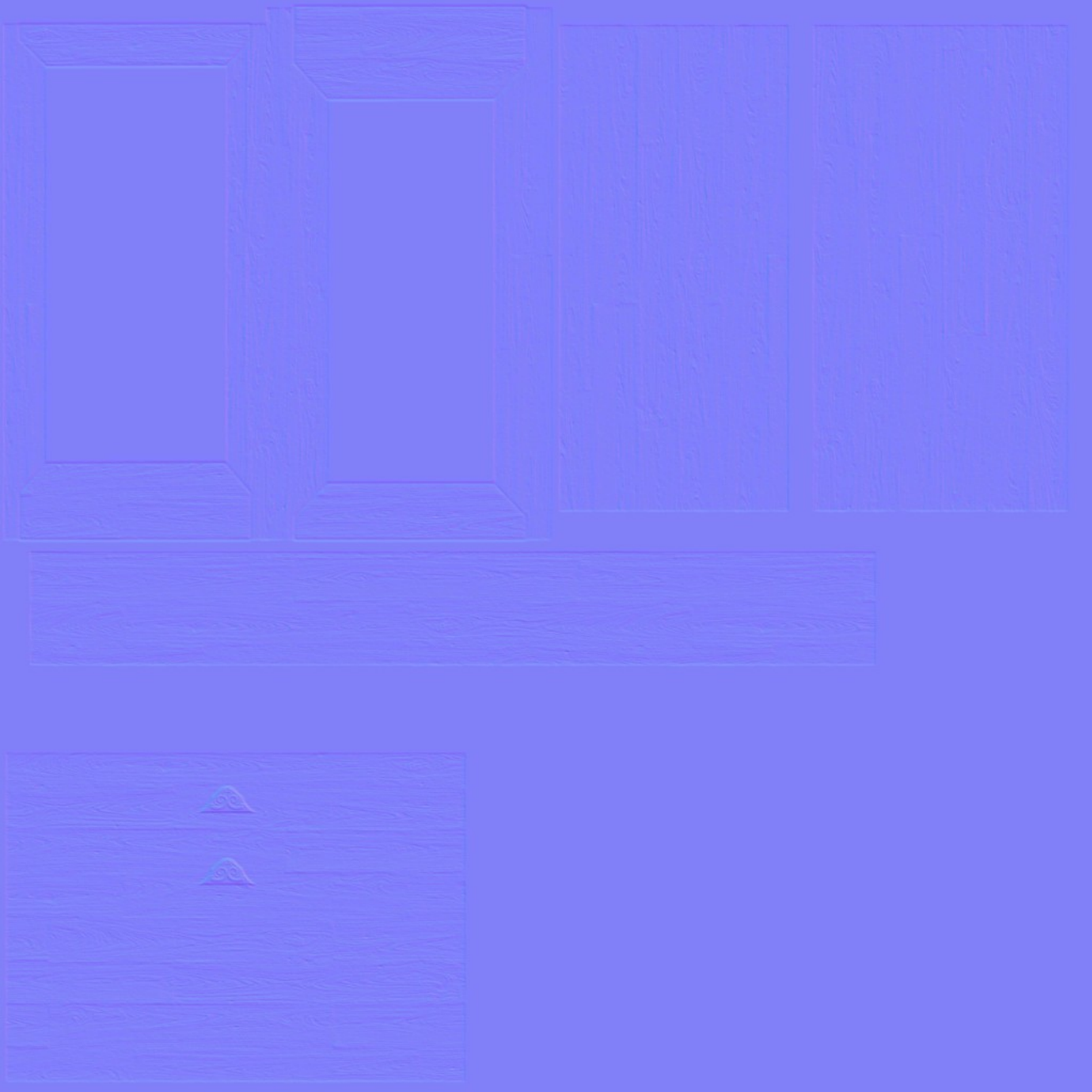
FAKULTA STROJNÍ

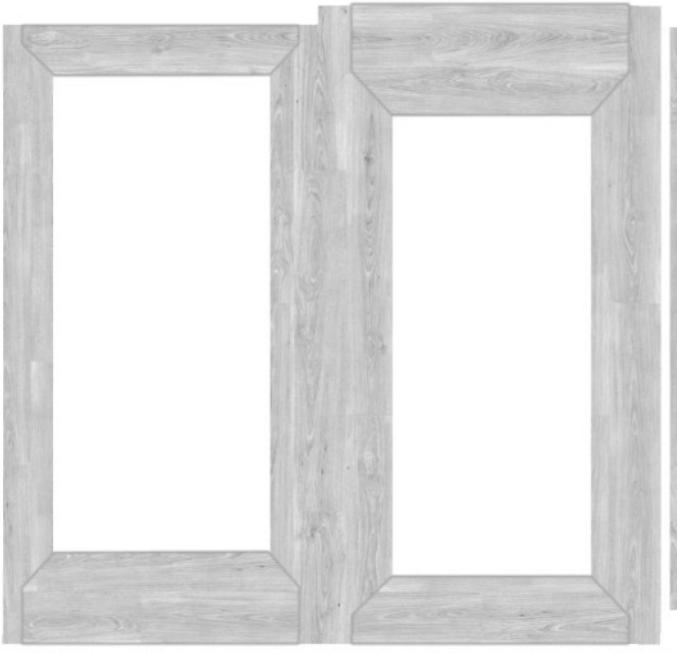
FAKULTA STROJNÍ

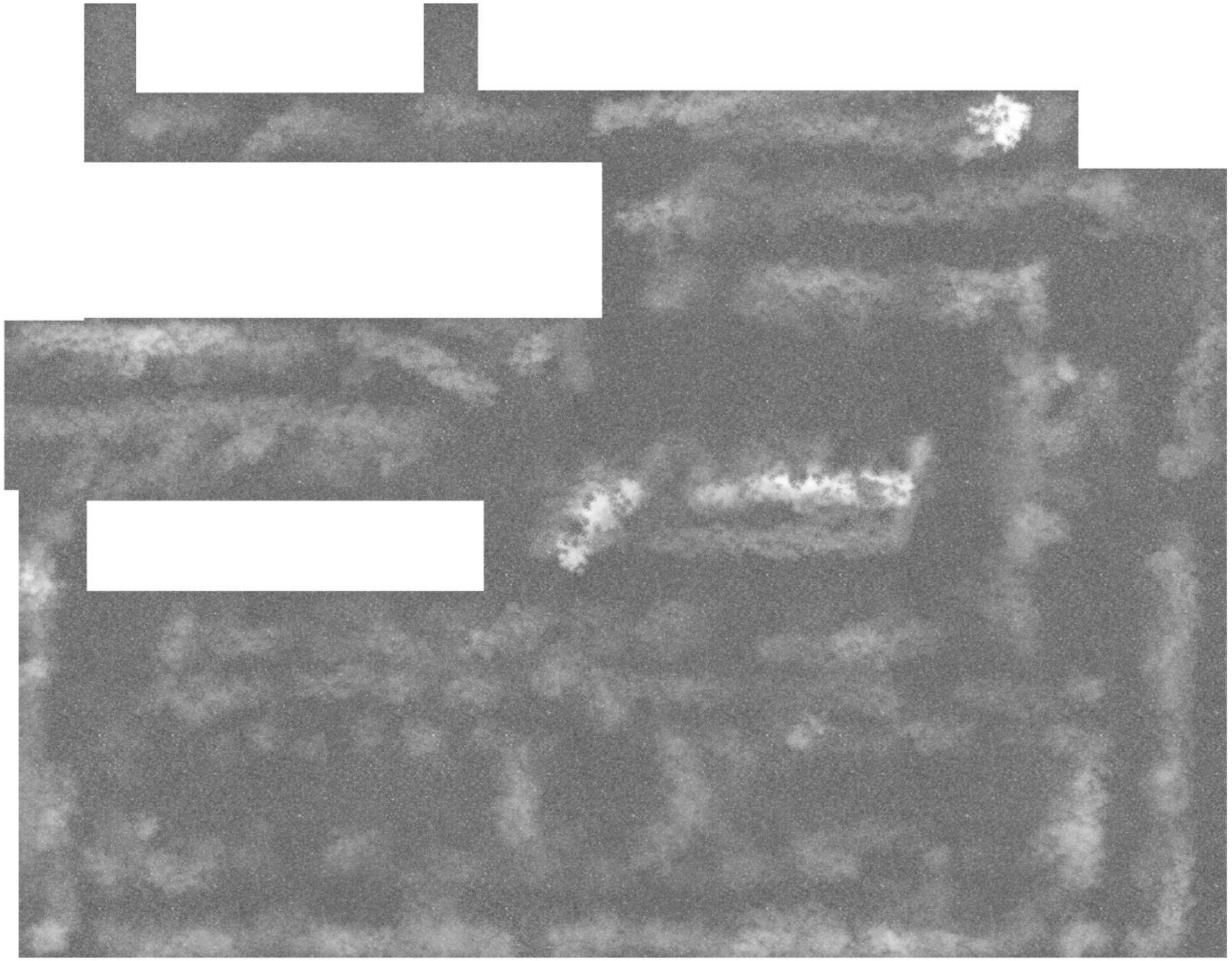

ELEKTRICKÉ ZAŘÍZENÍ
Neotvírat ochranný poklop

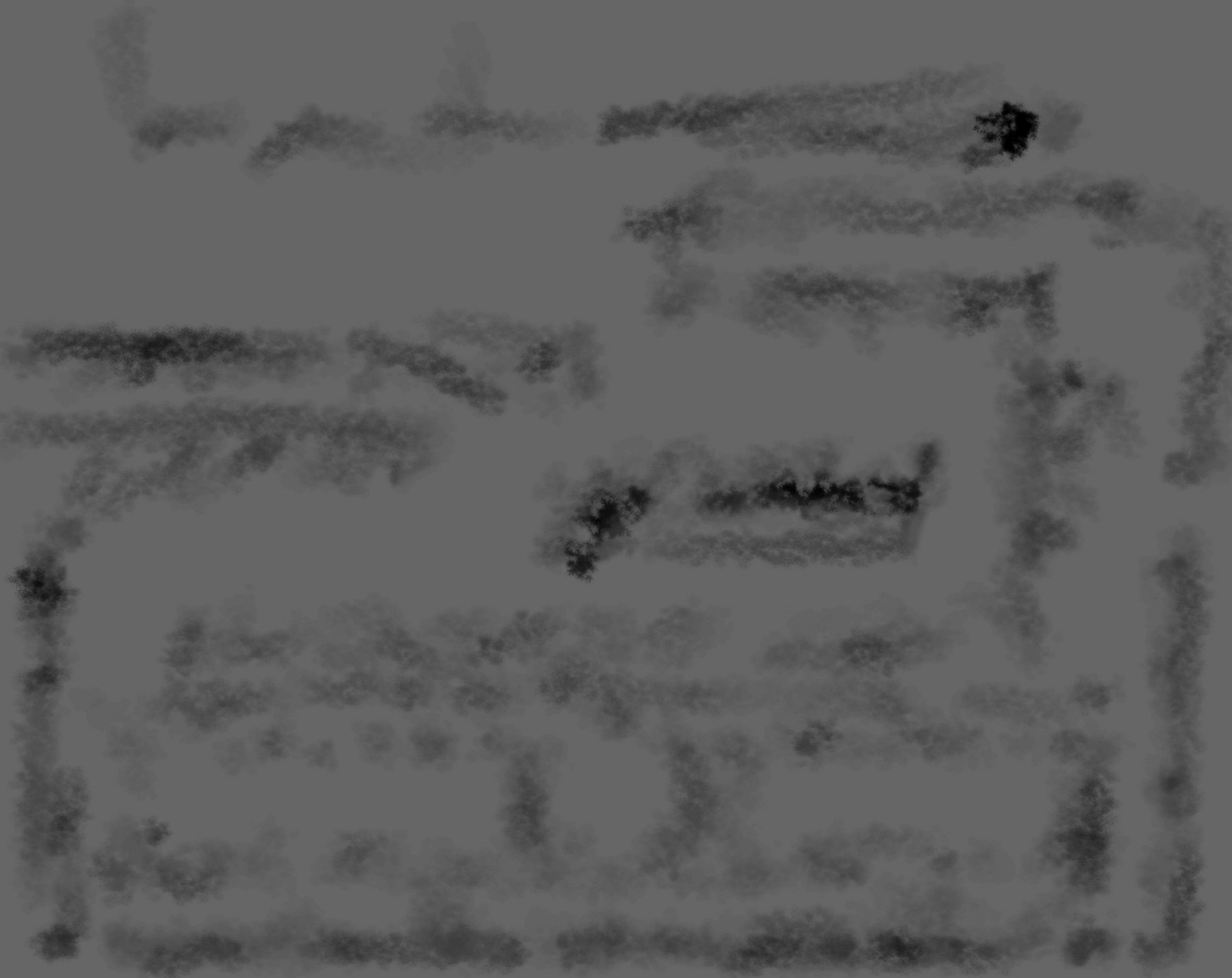


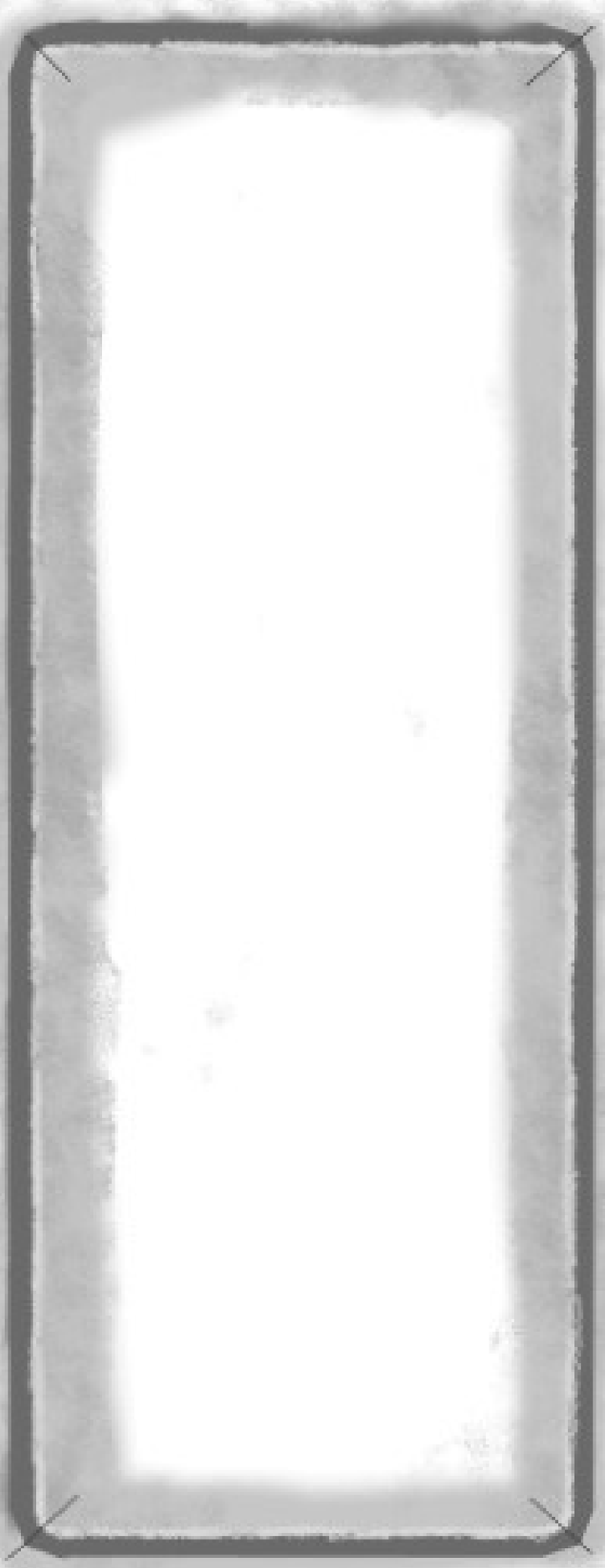


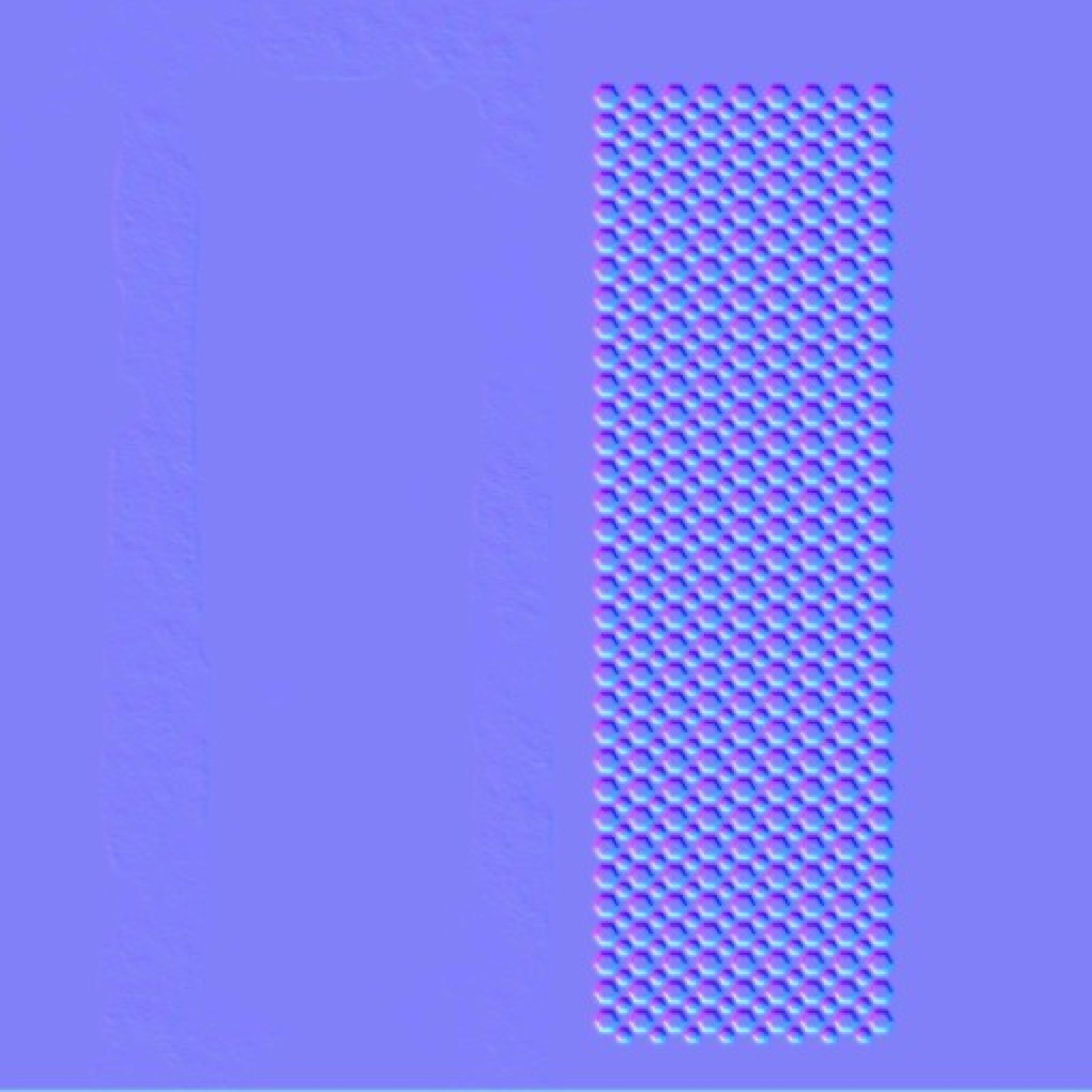



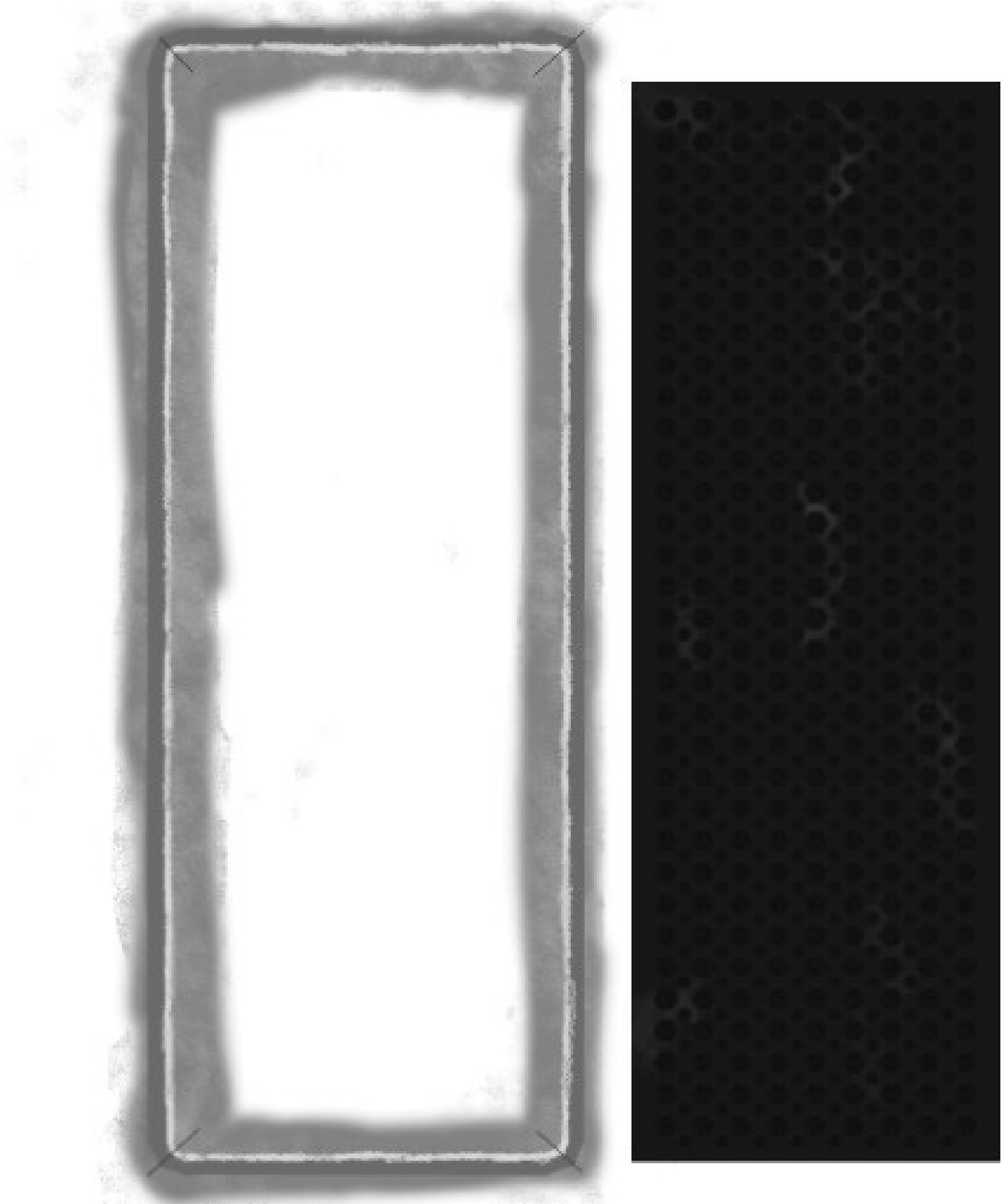


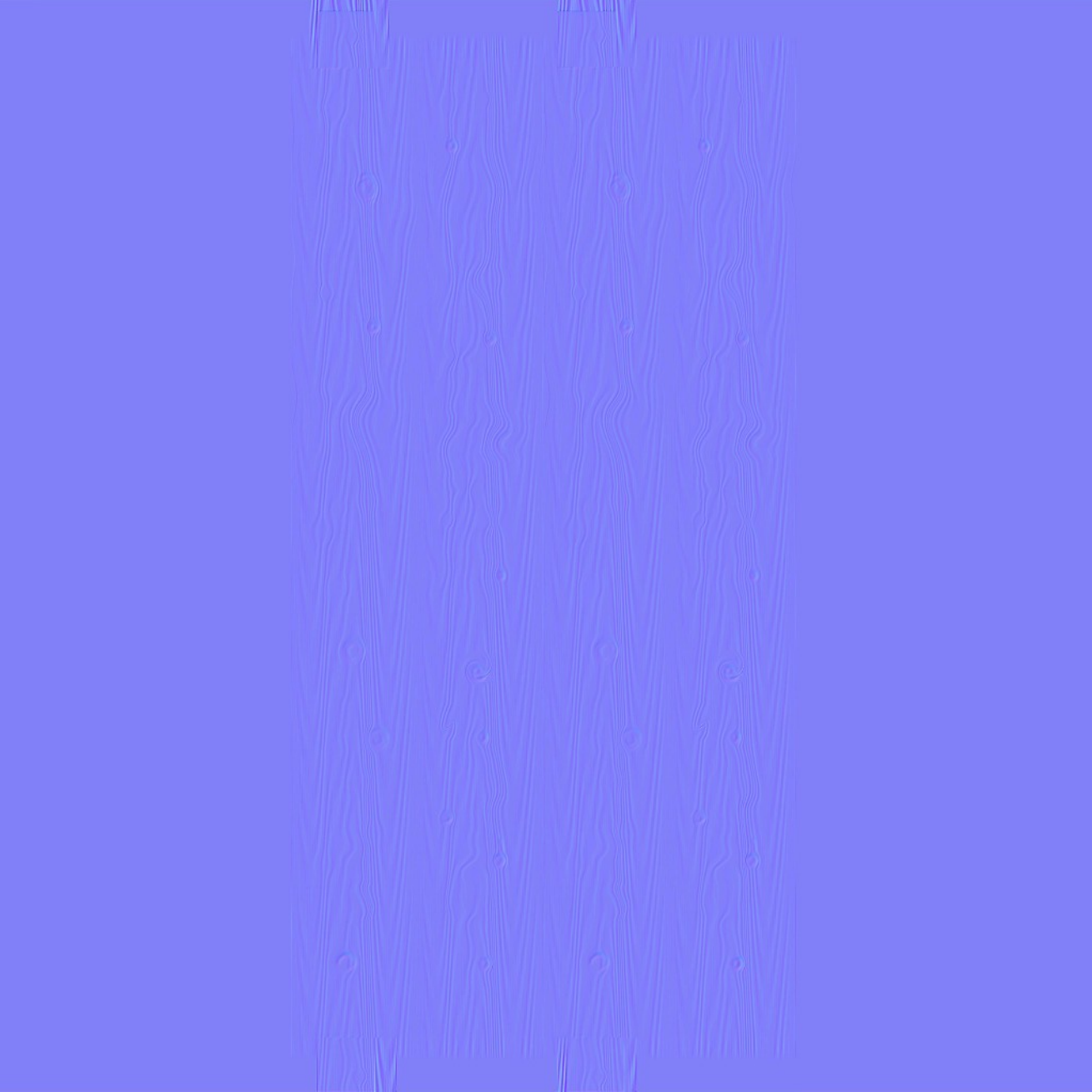


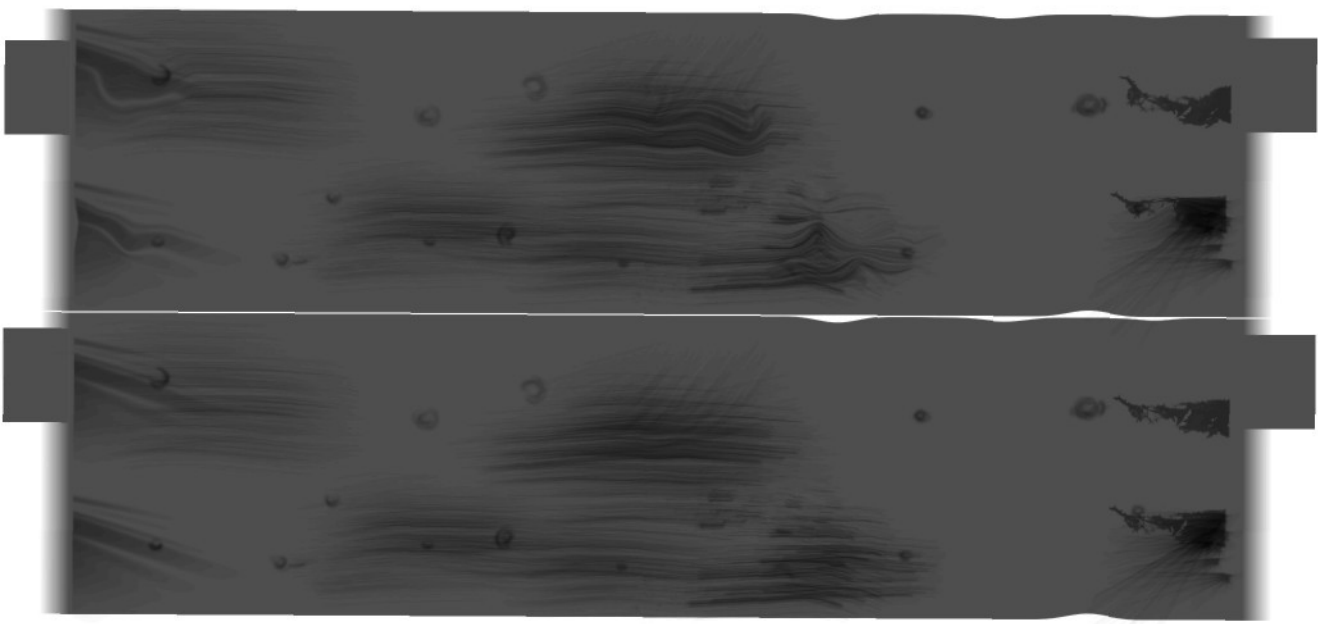


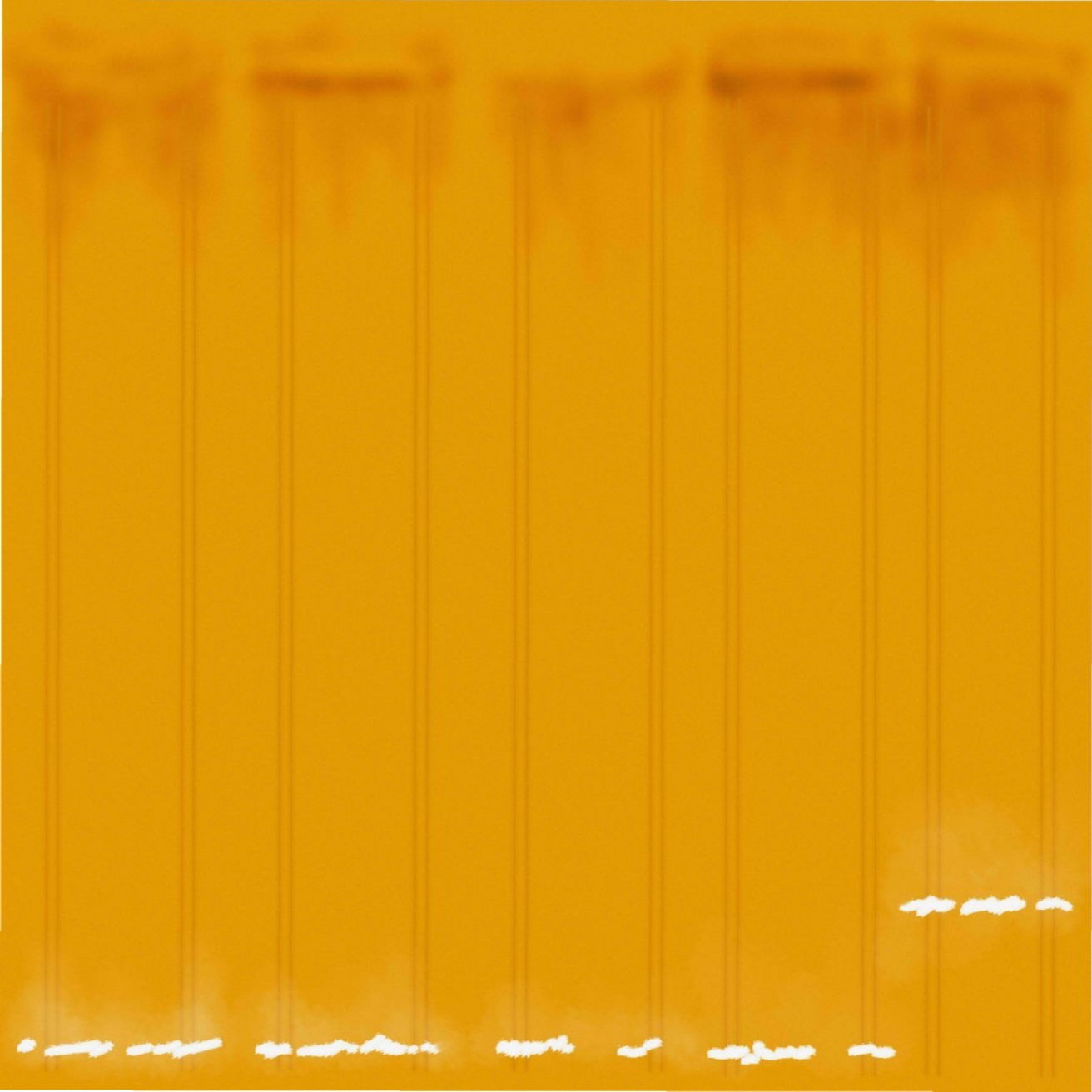


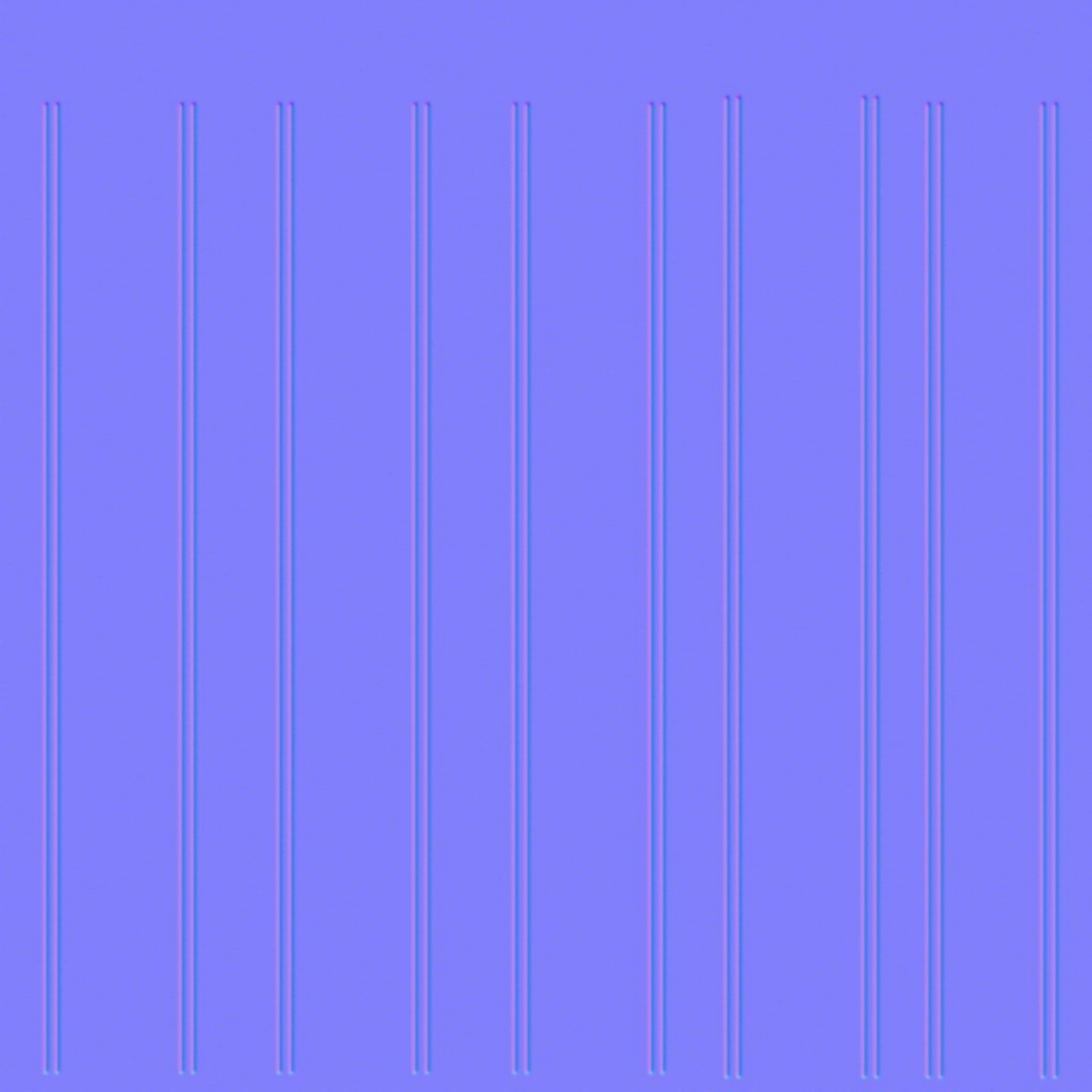


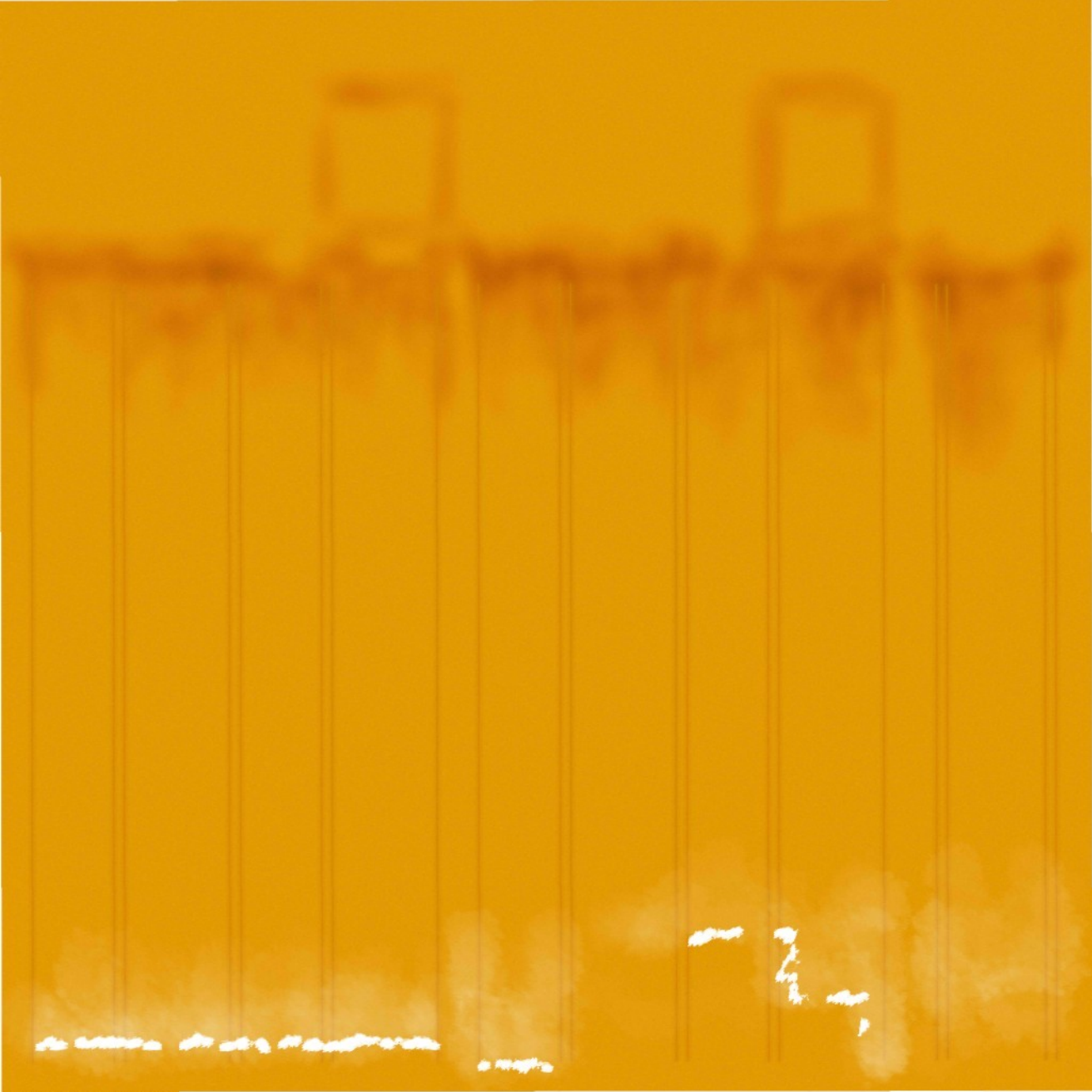


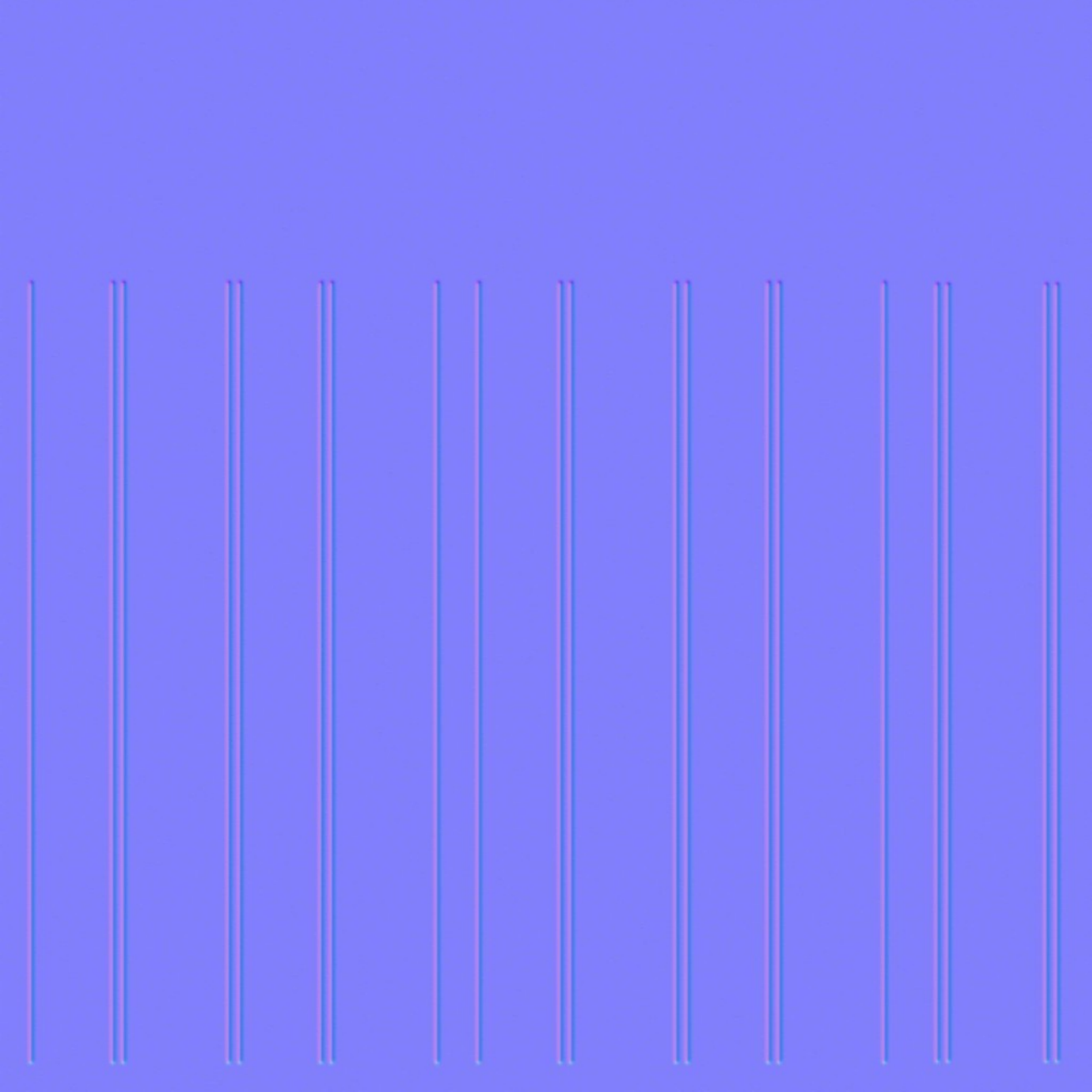


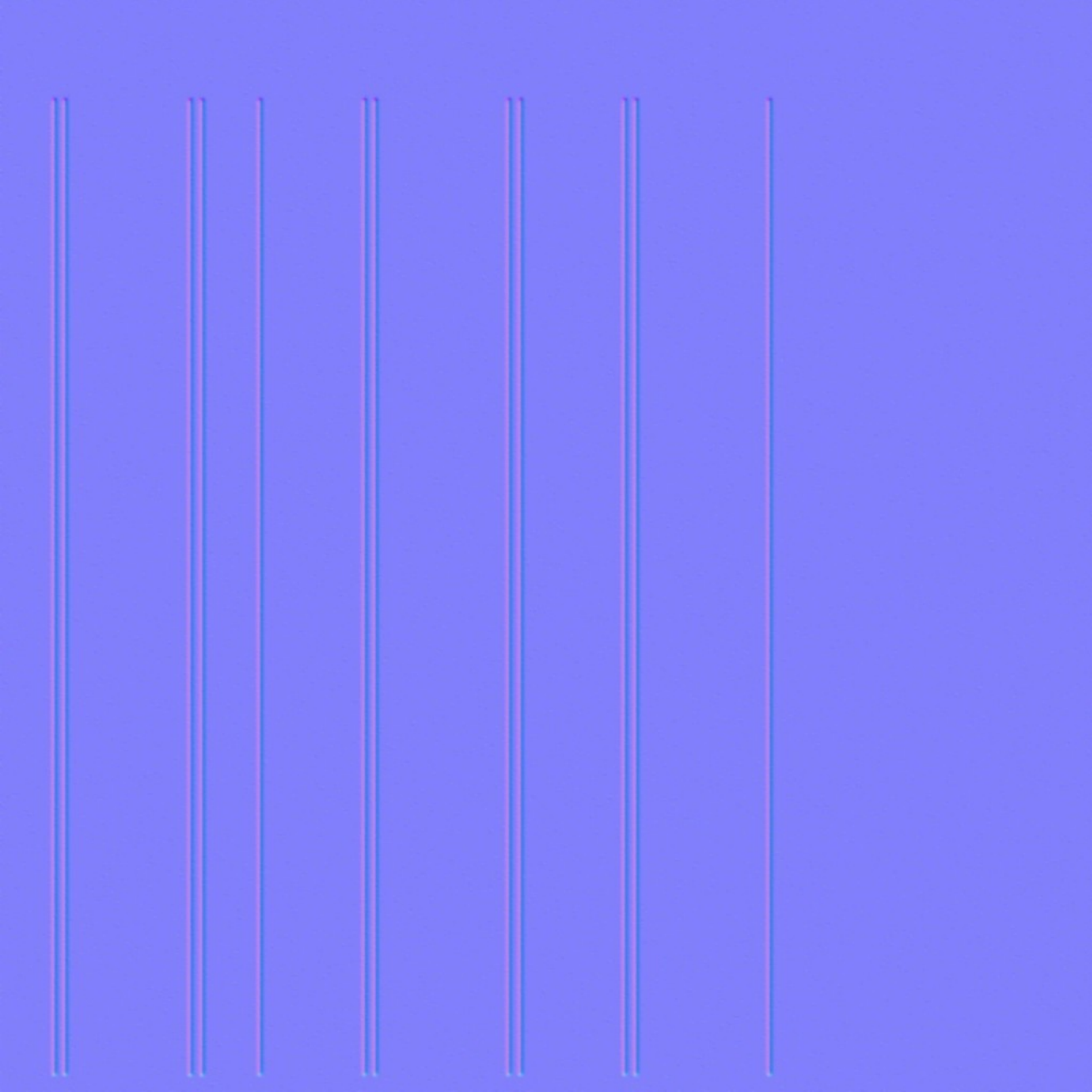


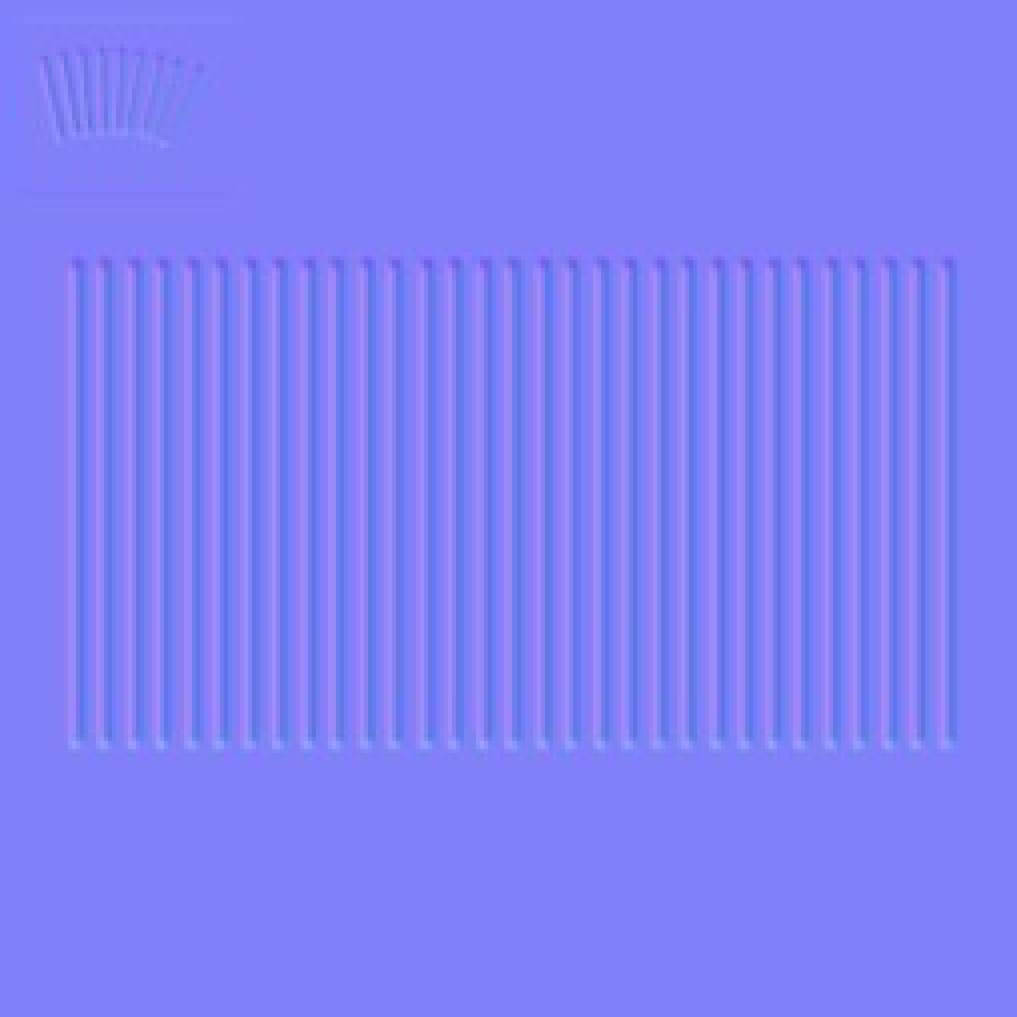


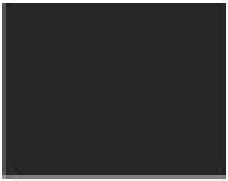


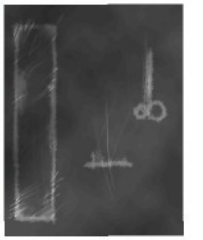












Obecně závazná vyhláška č. 2/2008,

kterou se mění a doplňuje vyhláška č. 4/2006 o místním poplatku za užívání veřejného prostranství

Zastupitelstvo města Chrastavy se na svém zasedání dne 14. dubna 2008 usnesením č. 2008/03/XIII usneslo vydat na základě § 14 odst. 2 zákona č. 565/1990 Sb., o místních poplatcích, ve znění pozdějších předpisů (dále jen „zákon") a v souladu s § 84 písm. d) zákona č. 128/2000 Sb., o obcích (obecní zřízení), ve znění pozdějších předpisů a § 84 odst. 2 písm. h) zákona č. 128/2000 Sb., o obcích (obecní zřízení), tuto obecně závaznou vyhlášku:

Čl. I

Obecně závazná vyhláška č. 4/2006 ve znění vyhlášky č. 1/2007, o místním poplatku za užívání veřejného prostranství se mění a doplňuje takto:

Čl. 4, odst. 1 se mění a doplňuje takto:

Sazba poplatku

Sazba poplatku činí za každý i započatý m² a každý i započatý den:

a) za umístění dočasných staveb a zařízení sloužících pro poskytování služeb (stánky, prodejní prodejny, vozíky, boxy a jiná zařízení)
   náměstí T. Seiferta 100,– Kč
   ostatní veřejná prostranství 20,– Kč

b) za umístění dočasných staveb a zařízení sloužících pro poskytování prodeje (stánky, prodejní prodejny, vozíky, boxy a jiná zařízení)
   náměstí T. Seiferta 100,– Kč
   ostatní veřejná prostranství 20,– Kč

Čl. II
Účinnost

Tato obecně závazná vyhláška nabývá účinnosti patnáctým dnem po jejím vyhlášení.

Ing. Richard Czech
místostarosta

Mgr. Miloslav Pilař
místostarosta

Vyvěšeno dne: 22. 5. 2008
Sejmuto dne:


Technická univerzita v Liberci

POŽÁRNÍ POPLACHOVÉ SMĚRNICE

Budova A

1. [illegible]

Místní ohlašovna požáru
kl. 9 (v pracovní době)
kl. 2211 (v mimopracovní době)

2. [illegible]

„HOŘÍ“

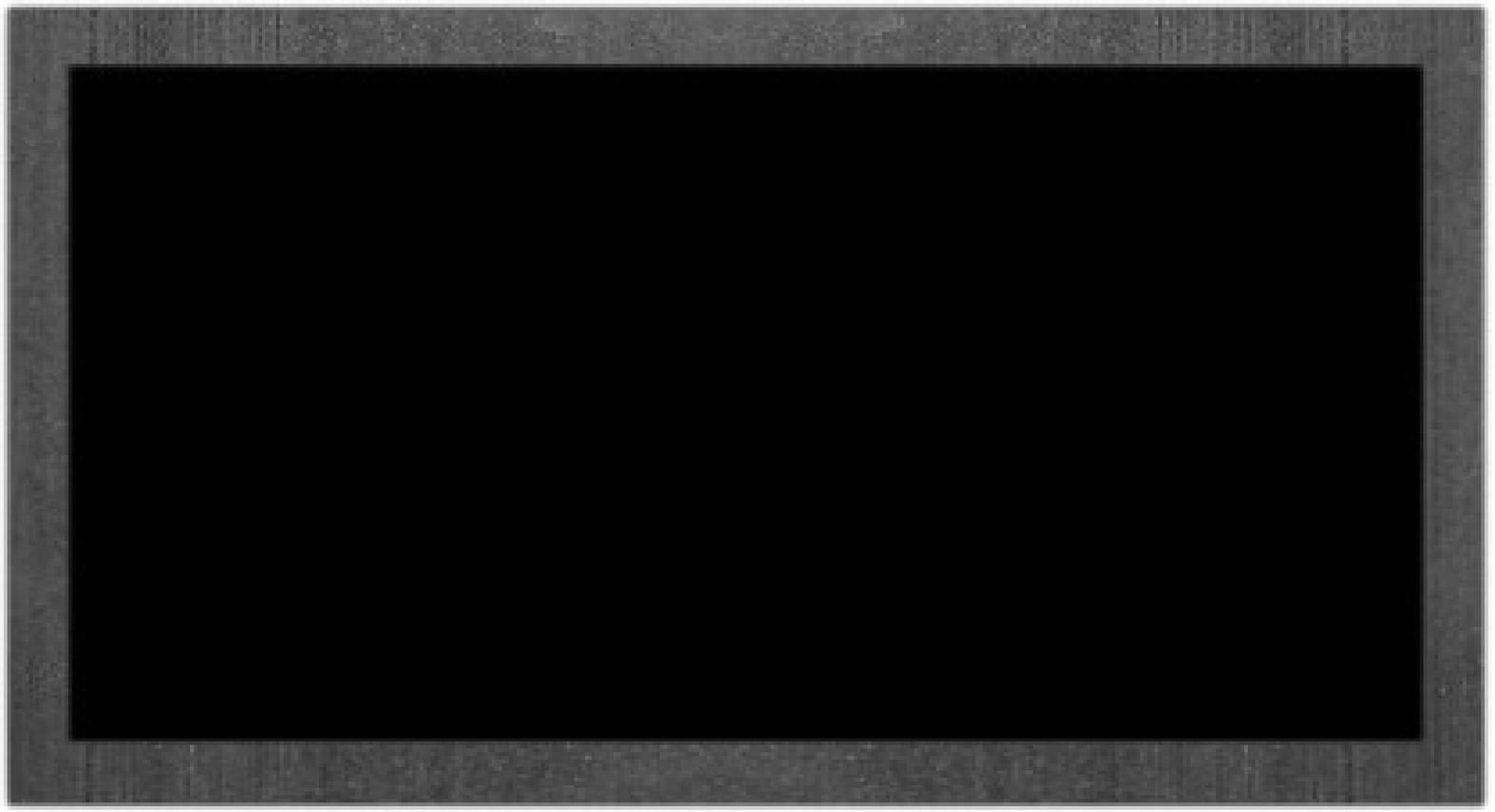
3. [illegible]
4. [illegible]
5. [illegible]

ZÁCHRANNÁ SLUŽBA	155	PORUCHY-VODA	[illegible]
POLICIE ČR	158	PORUCHY - EL. PROUD	[illegible]
TÍSŇOVÉ VOLÁNÍ	112	PORUCHY - PLYN	1239
		HASIČSKÝ ZÁCHRANNÝ SBOR LIBERECKÉHO KRAJE	[illegible]

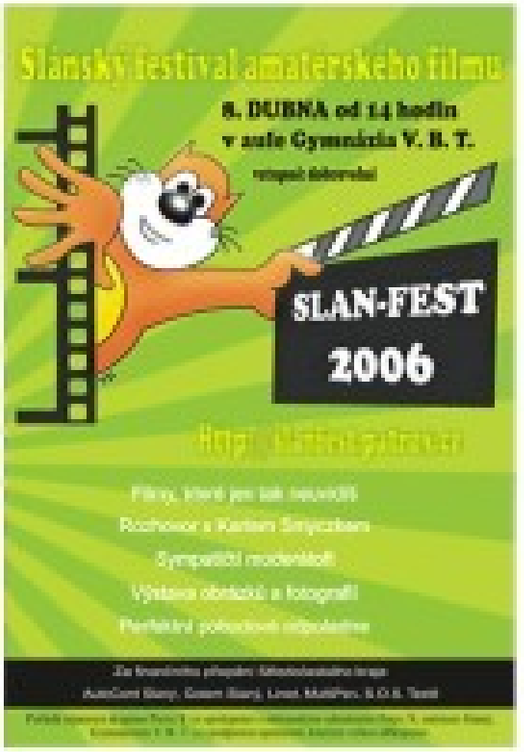
[illegible]

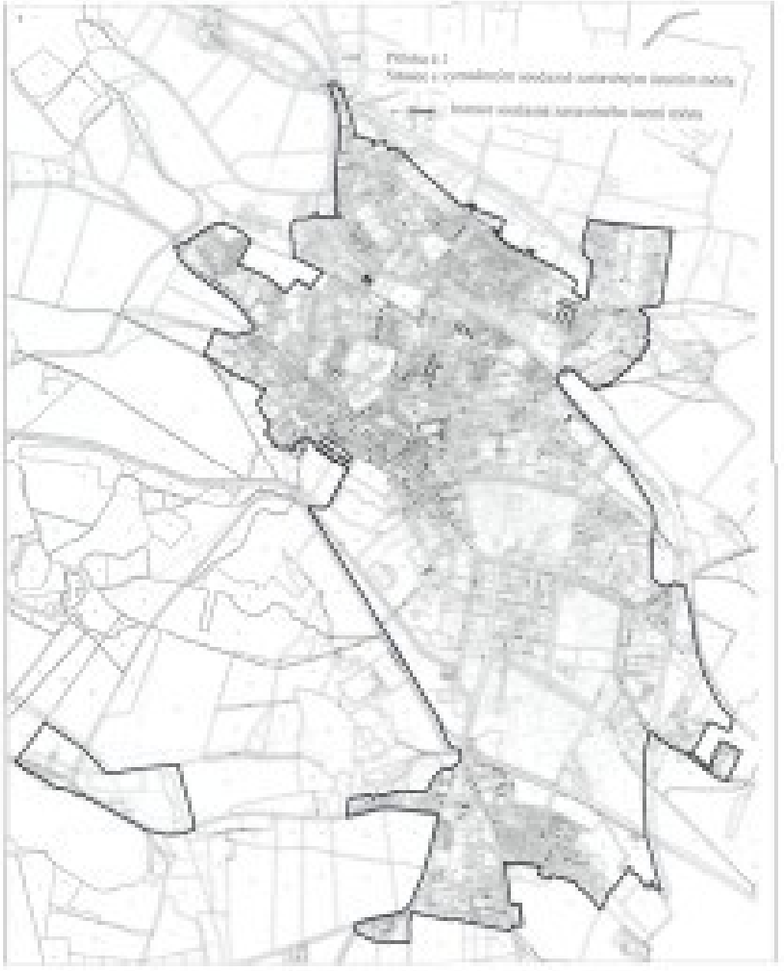
6. [illegible]


INFORMAČNÍ DESKA




PŘIHLÁŠKA KE STUDIU





Městský úřad Kuřim
Odbor investiční a regionálního rozvoje
Jungmannova 948, 664 34 Kuřim

Č.j.: MěÚ/IRR/04/06/OIR/Lu V Kuřimi dne: 17. 8. 2006
Vyřizuje: Ing. Lukášová Jana, tel.: 541 622 771

VEŘEJNÁ VYHLÁŠKA

o projednání koncepce regulačního plánu lokality NA ROVINÁCH pro obec Hvozdec

Pořizovatel regulačního plánu lokality NA ROVINÁCH pro obec Hvozdec – Městský úřad Kuřim, Odbor investiční a regionálního rozvoje, zahajuje v souladu s ustanovením § 21 zákona č. 50/1976 Sb., o územním plánování a stavebním řádu (dále jen stavební zákon), ve znění pozdějších předpisů a v souladu s vyhláškou č. 135/2001 Sb., o územně plánovacích podkladech a územně plánovací dokumentaci, ve znění pozdějších předpisů projednávání

koncepce regulačního plánu lokality NA ROVINÁCH pro obec Hvozdec.

V souladu s ustanovením § 21 stavebního zákona a § 52 vyhl. č. 135/2001 Sb. vyzývám tímto a projednání koncepce regulačního plánu dotčené orgány státní správy, orgány územního plánování sousedních územních obvodů, fyzické a právnické osoby, veřejnost.

Koncepce regulačního plánu bude vystavena k nahlédnutí

po dobu 30 dnů od 30.8. 2006 – 5. 10. 2006

- v pořizovatelně – na MěÚ Kuřim, odboru investiční a regionálního rozvoje, Jungmannova 948, Kuřim, 2. patro, chodba č. 300 (k nahlédnutí doporučujeme využít nejlépe úřední dny, tj. pondělí a středa od 7.30 – 11.30 a od 12.30 – 17.00 hodin,
- na Obecním úřadě v obci Hvozdec, č.p. 16 (k nahlédnutí doporučujeme využít nejlépe úřední dny, tj. pondělí od 7.30 – 12.00 a od 12.45 – 18.00 hod a čtvrtek od 7.30 – 12.00 a od 12.45 – 17.00 hodin).

Koncepce regulačního plánu obsahuje textovou a výkresovou část. Pro orientaci v území může posloužit schválený Územní plán sídelního útvaru Hvozdec, který je k nahlédnutí na Městském úřadu Oblastního úřadu Veverská Bítýška a na Obecním úřadu obce Hvozdec.

Veřejné projednání koncepce regulačního plánu lokality NA ROVINÁCH s odborným výkladem se uskuteční dne 4. 10. 2006 (středa) v 18.30 hod. na Obecním úřadu v obci Hvozdec.

V souladu s § 21 stavebního zákona může dotčené orgány státní správy svá stanoviska písemnou formou do 30 dnů ode dne veřejného jednání. Do 30 dnů ode dne veřejného projednání může každý uplatnit své připomínky. Vlastníci pozemků a staveb, jejichž práva jsou dotčena koncepcí řešení regulačního plánu mohou podat své námitky do 30 dnů od veřejného projednání písemně na odbor:

MěÚ Kuřim, Odbor investiční a regionálního rozvoje, Jungmannova 948, Kuřim 664 34.

Ke stanovisku podanému po této lhůtě se nepřihlíží.

MĚSTSKÝ ÚŘAD KUŘIM
Jungmannova 948
664 34 Kuřim

Ing. arch. Ludmila Kubanová
vedoucí odboru

Oznámení vyvěšeno dne: 30.8. 2006
sejmuto dne:

IČ: 002 63 044
Bankovní spojení: KB Brno-město
K.ú. a č. jednotlivých účtů: 19-52824-441/0100

Tel.: 541 622 111, Fax: 548 259 843
El. pošta: posta@mukurim.cz
www.kurim.cz

FINANCNI URAD V TEPLICICH

[illegible]

V Teplicích 25.05.2006

VEŘEJNÁ VYHLÁŠKA

ve smyslu ustanovení § 19 odst. 1 zákona č. 337/1992 Sb. o správě daní a poplatků, ve znění pozdějších předpisů

Finanční úřad v Teplicích oznamuje Vám/jsem rozhodnou úřední písemnost pro dlužný subjekt:

Jana Weberová, rok nar.1965

bydliště trvalý bydliště: Mediany 34, Modřany 417 13

(bydliště písemností): 7221/108,[10960/1481]

(a pro bydliště zahájení adrese pro doručování Sobětínova 1825, Teplice 415 01, si dlužný subjekt nepřevzal nevyzvedl).

Písemnost jsou uložena po dobu vyvěšení veřejné vyhlášky (tj. po dobu patnácti dnů) na Finančním úřadě v Teplicích, Dlouhá 42, kancelář č. 184 , 1 . poschodí.

Jméno vyvěšení veřejné vyhlášky: [illegible]

Jméno sejmutí veřejné vyhlášky: [illegible]

Ing.Marcela Kubíková
vedoucí oddělení majetkových daní a správních řízení

MATICE SLEZSKÁ, místní odbor Dolní Lomná

srdečně zve širokou veřejnost na tradiční

XXXVIII. SLEZSKÉ DNY DOLNÍ LOMNÁ

8. - 9. 9. 2007

po oba dny vystoupí soubory na dvou pódiích
v sobotu od 15.00 hod. * * * v neděli od 10.00 hod.

Hanička - Moravsko
Antošovice - Rusko
Poráčan - Slovensko
Orovan - Slovensko

sobota 8. 9. 2007

19.00 - 24.00 hod.
Lidová veselice při dechovce ze Slovácka - Strahanka

19.00 - 02.00 hod.
Taneční zábava pro náladu kraje skupina Bár

Starý DP Mánesovou stodolou za Sýkořín, Trnavu, Hávní, Mlandí a Jablunkovem a Čadcu

Po oba dny v areálu: ukázky výrobků lidových řemesel a ukázky krojů

Součástí: Krytá/otevřená galerie řemesel - Umělci Těšínského Slezska (obrazy, keramika)

Od sobotního OD dobrovolných hasičů z celé východní Moravskoslezské oblasti

SRDEČNĚ ZVEME • ÚPRIMNE POZÝVAME • SERDECHNO ZAPRASZAMY

Hlavní sponzoři: Obecní knihovna v Dolní Lomné • Místní Národní výbor • Místní odbor Matice slezské • Ochotnický soubor Slezská nit • Sbor dobrovolných hasičů • Místní Sbor dobrovolných hasičů

Hlavní sponzor: HORNÍ SUCHÁ, TECHNOLOGIE, akciová společnost

Obecně závazná vyhláška č. 1/2008,

kterou se mění a doplňuje vyhláška č. 6/2006 o místním poplatku za užívání veřejného prostranství

Zastupitelstvo města Chrastavy se na svém zasedání dne 14. února 2008 usnesením č. 2008/03/XII usneslo vydat na základě § 14 odst. 2 zákona č. 565/1990 Sb., o místních poplatcích, ve znění pozdějších předpisů (dále jen „zákon") a v souladu s § 10 písm. d) zákona č. 128/2000 Sb., o obcích (obecní zřízení), ve znění pozdějších předpisů a § 84 odst. 2 písm. h) zákona č. 128/2000 Sb., o obcích (obecní zřízení), tuto obecně závaznou vyhlášku:

Čl. 1

Obecně závazná vyhláška č. 6/2006 ve znění vyhlášky č. 1/2007, o místním poplatku za užívání veřejného prostranství se mění a doplňuje takto:

Čl. 6 odst. 1 se mění a doplňuje takto:

Sazba poplatku

Sazba poplatku činí za každý i započatý m² a každý i započatý den:

a) za umístění dočasných staveb a zařízení sloužících pro poskytování služeb (stánky, pojízdné prodejny, vozíky, boxy a jim podobné)
- letní období
- mimo letní období

b) za umístění dočasných staveb a zařízení sloužících pro poskytování prodeje (stánky, pojízdné prodejny, vozíky, boxy a jim podobné)
- letní období
- mimo letní období

Čl. 2
Účinnost

Tato obecně závazná vyhláška nabývá účinnosti patnáctým dnem po jejím vyhlášení.

Ing. Michael Czorn
starosta

Miloslav Filip
místostarosta

Vyvěšeno dne 27. 2. 2008
Sejmuto dne

Správa LIANE

univerzitní přihlašovací systém

nějaký text pěkně nečitelný na renderu:D

8:00 - 12:00	8:00 - 12:00
8:00 - 12:00	8:00 - 12:00
8:00 - 12:00	8:00 - 12:00
8:00 - 12:00	8:00 - 12:00
8:00 - 12:00	8:00 - 12:00

Vyhláška o odpadech č. 1/2001
Obec Lipová

Čl. 1

Čl. 2

Čl. 3

Čl. 4