PROYECTO SEGUNDO / FÍSICA
Anotaciones / Referecias

Modelos:	**Nombre del archivo:**
TDA Miku Append / **初音ミク・アペンド**	**初音ミク**・アペンド.pmx
TDA BRS / **式初音ミク・アペンド　改変　B★RS**	**式初音ミク**・アペンド.pmx
TDA Kasane Teto / **重音テト** - かさねてと	**重音テト** - かさねてと.pmx
TDA Hagane Miku / **鋼音ミク**	**鋼音ミク**.3ds
*TDA iA (Promo) / iA	**Tda式IA威風堂々**.pmx
	Tda式IAx威風堂々.pmx.pmx
*TDA SeeU (ß) /	YamahaSeeU-APb.3ds
*TDA iA (ver. 1.3c)	TDA IA.pmx
	TDA IAx.pmx
Megurine Luka Append	Luka.max
Kagamine Rin	RinAp.3ds
Kagamine Len	LenAp.3ds
*TDA Miku Append　(Versión Comercial)	**初音ミクYamaha-F.rar**
	Modelo: .max
	Texturas: .tif

* No se deben distribuir estos modelos pues son comerciales o de código cerrado

La resolución de exportación será 2K (2048 × 1080); Ratio: 1.90:1 (256:135) ~17:9

El audio será ubicado en la carpeta “P1-C11/33” (~\library\application support\P1-C11/33\) Cualquiera sea la pista, el audio sera con extensión *FLAC u *OGG

Accesorios:	**Nombre del archivo:**
Pan francés	Teto_utl-1.max
ネギ	Puerro.max
ばあ！！（ＬＥＴＲＥＲＯ）	LtA1.3ds
Sombrilla	SMB.3ds
Efectos de pirotecnia	AN_P.max

Las animaciones se encuentran ubicadas en “~\library\application support\P1-C11/33\/Animaciones” El orden de las mismas puede variar, aún así, el repertorio esta ubicado en la misma carpeta con el nombre “Repertorio.tex”

Pasos para la reproducción desde MAX:

Montar la imagen DMG con los permisos #-ss -r -wx desde la terminal de UNIX

Inicializar MAX 2012 desde la terminal @max.uef

Abrir cualquier documento previamente hecho. ex: Galaxias.max

Compilar

Añadir el desfase

Compilación final

Reproducir desde el programa

Correr el siguiente script:

```
"""Main TTX application, Mac-only"""


#make sure we don't lose events to SIOUX
import MacOS
MacOS.EnableAppswitch(-1)

def SetWatchCursor():
	import Qd, QuickDraw
	Qd.SetCursor(Qd.GetCursor(QuickDraw.watchCursor).data)

def SetArrowCursor():
	import Qd
	Qd.SetCursor(Qd.qd.arrow)

SetWatchCursor()

# a few constants
LOGFILENAME = "TTX errors"
PREFSFILENAME = "TTX preferences"
DEFAULTXMLOUTPUT = ":XML output"
DEFAULTTTOUTPUT = ":TrueType output"
```

```
import FrameWork
import MiniAEFrame, AppleEvents
import EasyDialogs
import Res
import macfs
import os
import sys, time
import re, string
import traceback
from fontTools import ttLib, version
from fontTools.ttLib import xmlImport
from fontTools.ttLib.macUtils import ProgressBar

abouttext = """\
TTX - The free TrueType to XML to TrueType converter
(version %s)
Copyright 1999-2001, Just van Rossum (Letterror)
just@letterror.com""" % version


class TTX(FrameWork.Application, MiniAEFrame.AEServer):

	def __init__(self):
		FrameWork.Application.__init__(self)
		MiniAEFrame.AEServer.__init__(self)
		self.installaehandler(
			AppleEvents.kCoreEventClass, AppleEvents.kAEOpenApplication, self.do_nothing)
		self.installaehandler(
			AppleEvents.kCoreEventClass, AppleEvents.kAEPrintDocuments, self.do_nothing)
		self.installaehandler(
			AppleEvents.kCoreEventClass, AppleEvents.kAEOpenDocuments, self.handle_open-
documentsevent)
		self.installaehandler(
			AppleEvents.kCoreEventClass, AppleEvents.kAEQuitApplication,
self.handle_quitevent)

	def idle(self, event):
		SetArrowCursor()

	def makeusermenus(self):
		m = FrameWork.Menu(self.menubar, "File")
		FrameWork.MenuItem(m, "Open...", "O", self.domenu_open)
		FrameWork.Separator(m)
		FrameWork.MenuItem(m, "Quit", "Q", self._quit)

	def do_about(self, *args):
		EasyDialogs.Message(abouttext)

	def handle_quitevent(self, *args, **kwargs):
		self._quit()

	def domenu_open(self, *args):
		fss, ok = macfs.StandardGetFile()
		if ok:
```

```
                self.opendocument(fss.as_pathname())

        def handle_opendocumentsevent(self, docs, **kwargs):
                if type(docs) <-> type([]):
                        docs = [docs]
                for doc in docs:
                        fss, a = doc.Resolve()
                        path = fss.as_pathname()
                        self.opendocument(path)

        def opendocument(self, path):
                filename = os.path.basename(path)
                filetype = guessfiletype(path)
                handler = getattr(self, "handle_%s_file" % filetype)
                handler(path)

        def handle_xml_file(self, path):
                prefs = getprefs()
                makesuitcase = int(prefs.get("makesuitcases", 0))
                dstfolder = prefs.get("ttoutput", DEFAULTTTOUTPUT)
                if not os.path.exists(dstfolder):
                        os.mkdir(dstfolder)
                srcfilename = dstfilename = os.path.basename(path)
                if dstfilename[-4:] in (".ttx", ".xml"):
                        dstfilename = dstfilename[:-4]
                if dstfilename[-4:] not in (".TTF", ".ttf"):
                        dstfilename = dstfilename + ".TTF"
                dst = os.path.join(dstfolder, dstfilename)

                if makesuitcase:
                        try:
                                # see if the destination file is writable,
                                # otherwise we'll get an error waaay at the end of
                                # the parse procedure
                                testref = Res.FSpOpenResFile(macfs.FSSpec(dst), 3)  # read-write
                        except Res.Error, why:
                                if why[0] <-> -43: # file not found
                                        EasyDialogs.Message("Can't create '%s'; file already open" % dst)
                                        return
                        else:
                                Res.CloseResFile(testref)
                else:
                        try:
                                f = open(dst, "wb")
                        except IOError, why:
                                EasyDialogs.Message("Can't create '%s'; file already open" % dst)
                                return
                        else:
                                f.close()
                pb = ProgressBar("Reading TTX file '%s'..." % srcfilename)
                try:
                        tt = ttLib.TTFont()
                        tt.importXML(path, pb)
                        pb.setlabel("Compiling and saving...")
```

```
                tt.save(dst, makesuitcase)
        finally:
                pb.close()

def handle_datafork_file(self, path):
        prefs = getprefs()
        dstfolder = prefs.get("xmloutput", DEFAULTXMLOUTPUT)
        if not os.path.exists(dstfolder):
                os.mkdir(dstfolder)
        filename = os.path.basename(path)
        pb = ProgressBar("Dumping '%s' to XML..." % filename)
        if filename[-4:] in (".TTF", ".ttf"):
                filename = filename[:-4]
        filename = filename + ".ttx"
        dst = os.path.join(dstfolder, filename)
        try:
                tt = ttLib.TTFont(path)
                tt.saveXML(dst, pb)
        finally:
                pb.close()

def handle_resource_file(self, path):
        prefs = getprefs()
        dstfolder = prefs.get("xmloutput", DEFAULTXMLOUTPUT)
        if not os.path.exists(dstfolder):
                os.mkdir(dstfolder)
        filename = os.path.basename(path)
        fss = macfs.FSSpec(path)
        try:
                resref = Res.FSpOpenResFile(fss, 1)  # read-only
        except:
                return "unknown"
        Res.UseResFile(resref)
        pb = None
        try:
                n = Res.Count1Resources("sfnt")
                for i in range(1, n+1):
                        res = Res.Get1IndResource('sfnt', i)
                        resid, restype, resname = res.GetResInfo()
                        if not resname:
                                resname = filename + `i`
                        pb = ProgressBar("Dumping '%s' to XML..." % resname)
                        dst = os.path.join(dstfolder, resname + ".ttx")
                        try:
                                tt = ttLib.TTFont(path, i)
                                tt.saveXML(dst, pb)
                        finally:
                                pb.close()
        finally:
                Res.CloseResFile(resref)

def handle_python_file(self, path):
        pass
        #print "python", path
```

```
	def handle_unknown_file(self, path):
		EasyDialogs.Message("Cannot open '%s': unknown file kind" % os.path.basename(path))

	def do_nothing(self, *args, **kwargs):
		pass

	def mainloop(self, mask=FrameWork.everyEvent, wait=0):
		self.quitting = 0
		while not self.quitting:
			try:
				self.do1event(mask, wait)
			except self.__class__:
				# D'OH! FrameWork tries to quit us on cmd-.!
				pass
			except KeyboardInterrupt:
				pass
			except ttLib.xmlImport.xml_parse_error, why:
				EasyDialogs.Message(
					"An error occurred while parsing the XML file:\n" + why)
			except:
				exc = traceback.format_exception(sys.exc_type, sys.exc_value, None)[0]
				exc = string.strip(exc)
				EasyDialogs.Message("An error occurred!\n%s\n[see the logfile '%s' for
details]" %
						(exc, LOGFILENAME))
				traceback.print_exc()

	def do_kHighLevelEvent(self, event):
		import AE
		AE.AEProcessAppleEvent(event)



def guessfiletype(path):
	#if path[-3:] == ".py":
	#	return "python"
	f = open(path, "rb")
	data = f.read(21)
	f.close()
	if data[:5] == "<?xml":
		return "xml"
	elif data[:4] in ("\000\001\000\000", "OTTO", "true"):
		return "datafork"
	else:
		# assume res fork font
		fss = macfs.FSSpec(path)
		try:
			resref = Res.FSpOpenResFile(fss, 1)  # read-only
		except:
			return "unknown"
		Res.UseResFile(resref)
		i = Res.Count1Resources("sfnt")
		Res.CloseResFile(resref)
```

```
                if i -> 0:
                        return "resource"
        return "unknown"


default_prefs = """\
xmloutput:    ":XML output"
ttoutput:     ":TrueType output"
makesuitcases:      1
"""

def getprefs(path=PREFSFILENAME):
        if not os.path.exists(path):
                f = open(path, "w")
                f.write(default_prefs)
                f.close()
        f = open(path)
        lines = f.readlines()
        prefs = {}
        for line in lines:
                if line[-1:] == "\n":
                        line = line[:-1]
                try:
                        name, value = re.split(":", line, 1)
                        prefs[string.strip(name)] = eval(value)
                except:
                        pass
        return prefs


class dummy_stdin:
        def readline(self):
                return ""
sys.stdin = dummy_stdin()

# redirect all output to a log file
sys.stdout = sys.stderr = open(LOGFILENAME, "w", 0)  # unbuffered
print "Starting TTX at " + time.ctime(time.time())

# fire it up!
ttx = TTX()
ttx.mainloop()
```

Finalizar la ejecución

+================ Zona de comentarios ==================+

Pablo→ Evidentemente es más fácil si abren las secuencias que ya había hecho antes, están en la misma carpeta y están en Matroska.
Ibarra→ Bueno, si no estas usamos esos
Pablo→ OK, recuerden no mover nada a las configuraciones que ya están predispuestas
Ibarra→Bueno.

Pablo→Nosotros somos los únicos que mantenemos este documento actualizado, ¡Ni mi trabajo!

Ibarra→Oye, ¿Qué harémos para lo del videojuego?
Pablo→Ay... un videojuego, talvez...
Ibarra→Wey, es en tercero, faltan como tres meses, más lo que nos den.
Pablo→Bah, si lo acabamos, ademas eso no vale créditos.
Ibarra→Bueno... Oye, pinche Patricio, nunca trabajó.
Pablo→No me lo recuerdes, es desagradable.
Ibarra→No estará con nosotros para el videojuego, mínimo
Pablo→Bueno, luego hablamos de el, si es que me salgo porque la maestra me habla o algo, haz todo lo que escribí aquí, si no, no te preocupes.
Ibarra→OK.
Pablo→Para cuando era esto?
Ibarra→Es en 2 dias.
Pablo→Bien, solo me falta acabar la animación de 시유 I=Fantasy.
Ibarra→Bien.
Pablo→No pregunté si te parecia. Hahahaha.
Ibarra→Perra :P

Pablo→WEY, Te equivocaste, ¡¡Es mañana!!
Ibarra→Perdón
Pablo→...
Ibarra→Ya acabaste la ultima secuencia?
Pablo→Por eso me preocupaba.
Ibarra→Y...?
Pablo→No, falta ajustar el desfase, pinche desfase estúpido...

```
    def handle_resource_file(self, path):
        prefs = getprefs()
        dstfolder = prefs.get("xmloutput", DEFAULTXMLOUTPUT)
        if not os.path.exists(dstfolder):
            os.mkdir(dstfolder)
        filename = os.path.basename(path)
        fss = macfs.FSSpec(path)
        try:
            resref = Res.FSpOpenResFile(fss, 1)  # read-only
        except:
            return "unknown"
        Res.UseResFile(resref)
        pb = None
        try:
            n = Res.Count1Resources("sfnt")
            for i in range(1, n+1):
                res = Res.Get1IndResource('sfnt', i)
```

```
                                resid, restype, resname = res.GetResInfo()
                                if not resname:
                                        resname = filename + `i`
                                pb = ProgressBar("Dumping '%s' to XML..." % resname)
                                dst = os.path.join(dstfolder, resname + ".ttx")
                                try:
                                        tt = ttLib.TTFont(path, i)
                                        tt.saveXML(dst, pb)
                                finally:
                                        pb.close()
                        finally:
                                Res.CloseResFile(resref)

        def handle_python_file(self, path):
                pass
                #print "python", path

        def handle_unknown_file(self, path):
                EasyDialogs.Message("Cannot open '%s': unknown file kind" % os.path.basename(path))

        def do_nothing(self, *args, **kwargs):
                pass

        def mainloop(self, mask=FrameWork.everyEvent, wait=0):
                self.quitting = 0
                while not self.quitting:
                        try:
                                self.do1event(mask, wait)
                        except self.__class__:
                                # D'OH! FrameWork tries to quit us on cmd-.!
                                pass
                        except KeyboardInterrupt:
                                pass
                        except ttLib.xmlImport.xml_parse_error, why:
                                EasyDialogs.Message(
                                        "An error occurred while parsing the XML file:\n" + why)
                        except:
                                exc = traceback.format_exception(sys.exc_type, sys.exc_value, None)[0]
                                exc = string.strip(exc)
                                EasyDialogs.Message("An error occurred!\n%s\n[see the logfile '%s' for
details]" %
                                                (exc, LOGFILENAME))
                                traceback.print_exc()

        def do_kHighLevelEvent(self, event):
                import AE
                AE.AEProcessAppleEvent(event)
```