Songlib: rra support

written by: Song Li Buser

Revision Date: February 9, 2012

A sample program

Here is a typical program for manipulating an RRA file. This program clips the audio so that no value exceeds the preset level:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <assert.h>
#include <songlib/util.h>
#include <songlib/rra.h>

#define CLIP_LEVEL 3000

/* change PROGRAM_NAME and PROGRAM_VERSION appropriately */

char *PROGRAM_NAME = "clip";
char *PROGRAM_VERSION = "0.01";

static int clip = CLIP_LEVEL;
static int amplify = 32768 / CLIP_LEVEL;

static int processOptions(int,char **);

int
main(int argc,char **argv)
    {
    int argIndex = 1;

    int i,j;
    int amp;
    FILE *in,*out;
    RRA *h;

    argIndex = processOptions(argc,argv);

    if (argc-argIndex == 0)
        {
        in = stdin;
        out = stdout;
        }
    else if (argc-argIndex == 1)
        {
        in = OpenFile(argv[argIndex],"r");
        out = stdout;
        }
    else if (argc-argIndex == 2)
        {
        in = OpenFile(argv[argIndex],"r");
        out = OpenFile(argv[argIndex+1],"w");
        }
    else
        {
        printf("usage: clip -aN -cN [<input rra file> [<output rra file>]]\n");
        exit(-1);
        }

    h = newRRAHeader();

    readRRAHeader(in,h,0);

    writeRRAHeader(out,h,"modifiedBy: clip",0);

    for (i = 0; i < h->samples; ++i)
        for (j = 0; j < h->channels; ++j)
            {
            amp = readRRAAmplitude(in,out,h->bitsPerSample,0);
            if (amp < - clip)
                fprintf(out,"%d\n",- clip * amplify);
            else if (amp > clip)
                fprintf(out,"%d\n",clip * amplify);
            else
                fprintf(out,"%d\n",amp * amplify);
            }

    fclose(in);
    fclose(out);

    return 0;
    }

/* only -oXXX  or -o XXX options */

static int
processOptions(int argc, char **argv)
{
    int argIndex;
    int argUsed;
    int separateArg;
    char *arg;

    argIndex = 1;

    while (argIndex < argc && *argv[argIndex] == '-')
```

```
        {
        separateArg = 0;
        argUsed = 0;

        if (argv[argIndex][2] == '\0')
            {
            arg = argv[argIndex+1];
            separateArg = 1;
            }
        else
            arg = argv[argIndex]+2;

        switch (argv[argIndex][1])
            {
            case 'a':
                amplify = atof(arg);
                argUsed = 1;
                break;
            case 'c':
                clip = atoi(arg);
                argUsed = 1;
                break;
            case 'h':
                printf("clip options:\n");
                printf("  -a N   set amplification level to N\n");
                printf("         default value is %d\n",amplify);
                printf("  -c N   set clip level to N\n");
                printf("         default value is %d\n",clip);
                printf("  -h     help\n");
                exit(0);
                break;
            case 'v':
                printf("%s version %s\n", PROGRAM_NAME, PROGRAM_VERSION);
                exit(0);
                break;
            default:
                Fatal("option %s not understood\n",argv[argIndex]);
            }

        if (separateArg && argUsed)
            ++argIndex;

        ++argIndex;
        }

    return argIndex;
    }
```

One might use this program to create a set of fuzzy guitar notes from a set of clean notes:

```
slbuser@songlib:~/notes$ for s in clean*.rra
> do
> cat $s | clip -t1000 -a15 > out.rra
> mv out.rra $s
```

```
    > done
```

An analysis of the program

The first RRA-specific function to be called in the above program is:

```
    RRA *newRRAHeader(void);
```

This function returns an empty RRA object. We use the variable *h* to point to the newly created object:

```
    h = newRRAHeader();
```

The next RRA function called is:

```
    void readRRAHeader(FILE *,RRA *,void (*)(RRA *,char *,void *));
```

Its job is to read the header information from the input file and place it into the empty RRA object;

```
    readRRAHeader(in,h,0);
```

The last argument of the function is the handler for attributes that *readRRAHeader* doesn't recognize. A zero argument means that the default unknown attribute handler is used. These attribute value pairs are placed by the default handler in a field called *items*.

The next function called is:

```
    void writeRRAHeader(FILE *,RRA *,char *,void (*)(FILE *,RRA *));
```

This call begins the output of the transformed RRA file. In the program above, the call:

```
    writeRRAHeader(out,h,"modifiedBy: clip",0);
```

writes the header information in *h*, adds an additional *modifiedBy* field value, and indicates that any non-standard attribute-value pairs be written out by the default unknown attribute handler, as indicated by the zero value.

Finally, we have the read/write loop that processes the amplitude values in the input RRA file:

```
for (i = 0; i < h->samples; ++i)
    for (j = 0; j < h->channels; ++j)
        {
        amp = readRRAAmplitude(in,out,h->bitsPerSample,outputComment);
        if (amp < - clip)
            fprintf(out,"%d\n",- clip * amplify);
        else if (amp > clip)
            fprintf(out,"%d\n",clip * amplify);
        else
            fprintf(out,"%d\n",amp * amplify);
        }
```

The function:

```
(int) readRRAAmplitude(FILE *,FILE *,int,void (*)(FILE *,FILE *));
```

is used to read in RRA amplitude values. Since these values can be either integer or real and can contain comments, the *readRRAAmplitude* function is preferred over:

```
fscanf(in,"%d",&amp);
```

The first argument to the function is the input file pointer, the second is the output file pointer (used by the last argument), the third argument is the number of bits per sample, and the last argument is the comment handler. The number of bits per sample is used to convert a real amplitude (generally in the range from -1.0 to 1.0) to an integer value. In the example program, the actual call is:

```
amp = readRRAAmplitude(in,out,h->bitsPerSample,outputComment);
```

The *outputComment* function is used to immediately write any comments encountered to output. If one wished to change the clipping level on the fly, a custom comment handler would be passed instead of *outputComment*. This custom handler would read in the comment and, if it was meant for the clipping program, parse it and modify the clipping level. If the comment was not meant for the clipping program, the custom handler should write the comment to output.

There are other functions for manipulating RRAs. They are:

function	*purpose*
`RRA *newRRA(int,int,int,int)`	create/return an empty RRA object w
`RRA *readRRA(FILE *,void (*)(RRA *,char *,void *))`	read an RRA file/return an RRA obj
`void createRRAData(RRA *)`	adds a data portion to an RRA heade
`void clearRRAData(RRA *)`	set all sample values to zero – the arg
`voidfreeRRA(RRA *)`	free an RRA object, including the dat
`RRA *cloneRRA(RRA *,RRA_TAG *(*)(RRA_TAG *))`	clones an RRA object – the first argu